
Pós-Graduação em Ciência da Computação

Israel Bruno dos Santos Duarte

# Análise de Desempenho de Sistemas em Ambientes Virtualizados: um estudo de caso do SIG@

Universidade Federal de Pernambuco
ibsd@cin.ufpe.br
www.cin.ufpe.br/~ibsd

Recife
2023

Israel Bruno dos Santos Duarte

# Análise de Desempenho de Sistemas em Ambientes Virtualizados: um estudo de caso do SIG@

Trabalho apresentado ao Programa de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

***Área de Concentração***: *Redes de Computadores e Sistemas Distribuídos*
***Orientador***: *Andson Marreiros Balieiro*

Recife
2023

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Duarte, Israel Bruno dos Santos.
Análise de Desempenho de Sistemas em Ambientes Virtualizados: um estudo de caso do SIG@ / Israel Bruno dos Santos Duarte. - Recife, 2023.
52 : il., tab.

Orientador(a): Andson Marreiros Balieiro
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2023.

1. SIG@. 2. Hipervisor. 3. Avaliação de Desempenho. 4. Contêiner. 5. Virtualização. I. Balieiro, Andson Marreiros. (Orientação). II. Título.

000 CDD (22.ed.)

# AGRADECIMENTOS

A minha família, especialmente meus pais e irmãos, que sempre me apoiaram em todos os momentos.

A Marco e suas notas de revisão. Aos meus amigos do NTI pelo apoio técnico na execução dos experimentos descritos nesse documento. Em especial ao quadrado mágico: Rafael Santana, Domingos, Artur e Tarcísio.

E também ao meu orientador Andson Balieiro pelo apoio e chance de finalmente concluir o curso de Engenharia da Computação

*“One day I will find the right words, and they will be simple."*

*–Jack Kerouac*

# ABSTRACT

Since 2002, the Integrated Academic Management System (SIGA) has been maintained and used by the Federal University of Pernambuco (UFPE) to manage institutional processes. During this period, various versions of the system have been hosted on different physical and virtual hardware configurations, with hardware virtualization by hypervisors currently being employed. However, container-based virtualization is already included in future plans in which the available employment options are under analysis. Thus, this study aims at analyzing the comparative performance of SIGA running in the current virtual machine-based environment to that executed in containers. To do so, a production-like test environment was designed and subjected to a test load with volume and behavior analogous to the period of maximum system utilization, student enrollment, and the two deployment scenarios were analyzed. The results indicate that the containerization of the system can be adopted without noticeable impacts on the end-user and providing memory resource savings with a slight increase in the number of disk accesses.

**Keywords:** SIG@, Hypervisor, ESXI, Container, Virtualization

# RESUMO

Desde 2002, o Sistema Integrado de Gestão Acadêmica (SIGA) é mantido e utilizado pela Universidade Federal de Pernambuco (UFPE) para gerenciar os processos institucionais. Neste período as várias versões do sistema foram hospedadas sobre diferentes configurações de hardwares físicos e virtuais, sendo a virtualização de hardware por hipervisores a atualmente empregada. No entanto, a virtualização baseada em contêineres já está incluída nos planos de adoção futuros e encontra-se atualmente em fase de análise das opções disponíveis. Desta forma, este trabalho busca analisar o desempenho comparativo do SIGA no ambiente atual, baseado em máquinas virtuais, com o executado em contêineres. Para isso, um ambiente de testes similar ao de produção foi criado, submetido a uma carga de testes com volume e comportamento análogo ao período de máxima utilização do sistema, a matrícula e os dois cenários de implantação foram analisados. Os resultados obtidos indicam que a conteinerização do sistema pode ser adotada sem impactos perceptíveis para o usuário final e ainda gerar economia de recursos de memória com leve aumento no número de acessos ao disco.

**Palavras-chave:** SIG@, Hipervisor, Avaliação de Desempenho, Contêiner, Virtualização

# LISTA DE FIGURAS

# LISTA DE TABELAS

# LISTA DE ACRÔNIMOS

| | |
|---|---|
| **CPU** | Central Processing Unit |
| **DC** | Data Center |
| **HTTP** | Hypertext Transfer Protocol |
| **JVM** | Java Virtual Machine |
| **MVC** | Model-View-Controller |
| **RHEL** | Red Hat Enterprise Linux |
| **SGBDR** | Sistema Gerenciador de Bancos de Dados Relacional |
| **SI** | Sistema de Informações |
| **SIG@** | Sistema de Informações e Gestão Acadêmica |
| **SO** | Sistema Operacional |
| **SSD** | *Solid State Drive* |
| **STI** | Superintendência de Tecnologia da Informações |
| **UF** | Universidade Federal |
| **UFPE** | Universidade Federal de Pernambuco |
| **US EPA** | United States Environmental Protection Agency |
| **VM** | Máquina virtual- *Virtual Machine* |
| **VMM** | Virtual Machine Monitor |
| **VMs** | Máquinas virtuais |

# LISTA DE ALGORITMOS

# SUMÁRIO

# 1

# INTRODUÇÃO

Em um semestre típico, uma Universidade Federal (UF) do porte da Universidade Federal de Pernambuco (UFPE) pode ter cerca de cinquenta mil alunos ativos, sem contar com milhares de técnicos administrativos e docentes. Para auxiliar na administração das UFs, são utilizados Sistema de Informações (SI) capazes de coletar e gerenciar dados de vários subsistemas, apoiando as operações e tomada de decisões.

Neste contexto, desde 2003, a UFPE utiliza o SIG@ (2023), que gerencia os processos institucionais de ensino (graduação e pós-graduação), pesquisa, extensão, restaurante universitário, eleição de reitor e gestão de bens e pessoas. Em seus primeiros anos, o SIG@ foi alvo de fortes críticas relacionadas ao alto tempo de resposta no acesso e até mesmo indisponibilidade do sistema. Porém, por via de várias iterações de versão, correções de bugs e modernização do *Data Center (DC)*, melhoras significativas nos índices de satisfação dos usuários foram alcançadas Madeira (2018).

O sistema inicia sua segunda década de vida na versão 3.97.0.27, executando em um ambiente bem menos modesto que o servidor torre que o abrigou em suas primeiras versões. Desde meados de 2014, quando o DC da Superintendência de Tecnologia da Informações (STI) adotou a tecnologia de virtualização de hardware, o sistema ganhou flexibilidade, resiliência, segurança e facilidade no gerenciamento (Sahoo *et al.*, 2010).

Porém, tem-se uma questão fundamental para o bom desempenho da aplicação: qual o requisito físico necessário para o seu funcionamento ótimo? Apesar de ser a principal ferramenta da UFPE, o SIG@ não é único sistema hospedado no DC e o mau dimensionamento de recursos alocados ao sistema pode impactar negativamente o seu próprio desempenho, assim como o desempenho de outras aplicações que compartilham o hardware (Shirinbab & Lundberg, 2015).

Atualmente a STI faz o uso da virtualização de hardware com hipervisores, porém a virtualização ao nível de sistema operacional, conhecida como conteinerização, já faz partes dos planos e está em fase de análise. Dentre as soluções de conteinerização estudadas, a implementação híbrida, utilizada nos provedores de computação na nuvem que oferecem hospedagem de contêineres *al dhuraibi et al. (2017)*, na qual os contêineres rodam sobre máquinas virtuais, se destaca como opção viável para a atual infraestrutura, mas requer a avaliação da estrutura mais adequada da aplicação do SIG@ na decisão de se adotar ou não contêineres, ou permanecer no

uso de máquinas virtuais.

Neste aspecto, este trabalho propõe uma análise comparativa do desempenho do SIG@ implementado em dois cenários que utilizam conceitos de virtualização distintos. Um com a aplicação hospedada em máquinas virtuais que se utilizam de um ambiente com conectividade, armazenamento e processamento. Outro com a aplicação executando em contêineres hospedados em máquinas virtuais, isto é, uma camada extra de virtualização, gerando *overhead* no sistema. Resultados mostram que no primerio cenário 95,5% das requisições de matrícula foram atendidas com sucesso com tempo médio resposta de 2679 ms. Ao passo que no segundo cenário, alcançou-se 95,6% de atendimentos com tempo médio resposta de 2215 ms e um consumo menor de memória RAM, porém de CPU e rede similares.

## 1.1 OBJETIVOS

Este trabalho tem como objetivo analisar comparativamente o desempenho do SIG@ em cenários de virtualização distintos e assim dar suporte:

- ao dimensionamento de recursos adequados para o funcionamento ótimo do SIG@;
- a tomada de decisão sobre a tecnologia que resultará em melhor utilização de recursos públicos.

## 1.2 ESTRUTURA DO DOCUMENTO

Este trabalho está dividido em 5 capítulos e uma bibliografia. Na presente seção são apresentadas algumas das características gerais desse Trabalho de Conclusão de Curso, além de descrever sua organização. O capítulo 2 apresenta o referencial teórico, que contém toda fundamentação de base sobre o tema deste trabalho. O capítulo 3 apresenta os ambientes experimentais, suas especificações de hardware e software, a descrição dos testes , assim como as ferramentas de coleta e análise de dados. No capítulo 4, os resultados obtidos nos testes são apresentados e comentados. Finalizando, no capítulo 5 são apresentadas acerca do trabalho desenvolvido e as sugestões de trabalhos futuros.

# 2

# REFERENCIAL TEÓRICO

Neste capítulo serão apresentados a contextualização histórica da virtualização de hardware, conceitos de virtualização, hipervisor e conteinerização, bem como algumas ferramentas de software relevantes para este trabalho.

## 2.1 CONTEXTUALIZAÇÃO HISTÓRICA

O conceito de virtualização data do final da década de 60 quando a IBM introduziu o monitor de máquina virtual, também conhecido com hipervisor. Esta ferramenta é uma camada de abstração entre o hardware e o software, que desassocia o hardware físico do sistema operacional, possibilitando a existência de mais de uma Máquina virtual- *Virtual Machine* (VM) com diferentes sistemas operacionais e que compartilham os mesmo recursos de hardware. Esta técnica permitiu que *mainframes* caros, como o IBM S/360, fossem compartilhados por múltiplas aplicações sob a supervisão do hipervisor Rosenblum & Garfinkel (2005).

Anos mais tarde, com o surgimento dos sistemas operacionais multitarefas e do barateamento do hardware, a virtualização foi perdendo espaço, uma vez que as instituições conseguiam adquirir servidores que conseguiam suprir suas demandas e por um valor acessível. Porém, esta abordagem trouxe alguns problemas consigo, dentre eles a baixa eficiência energética. Em 2007, a *United States Environmental Protection Agency (US EPA)* descreve em seu *Report to Congress on Server and Data Center Energy Efficiency* que servidores na época possuíam taxa de utilização do processador entre 5-15% e mesmo sob estas condições tais servidores consumiam de 60-90% da energia máxima. O mesmo texto aponta a virtualização como uma das principais ferramentas no aumento da eficiência energética, uma vez que ela permite a consolidação de múltiplas máquinas virtuais no mesmo servidor físico, diminuindo o volume de hardware necessário e compensando carga extra de processamento associada a introdução do hipervisor (Jin *et al.* (2013)).

O ganho na eficiência dos recursos aliado as facilidades no gerenciamento das máquinas virtuais fizeram da virtualização por hipervisor a principal forma de virtualização nos *data centers* nas últimas décadas. Porém, nos últimos anos outro tipo de virtualização vem ganhando destaque ao conseguir melhorias consideráveis de desempenho, a virtualização a nivel Sistema

Operacional (SO), também conhecida como conteinerização(Felter *et al.* (2015)).

## 2.2 VIRTUALIZAÇÃO BASEADA EM HIPERVISOR

Segundo Sahoo *et al.* (2010), a virtualização é comumente definida como a tecnologia que introduz uma camada de abstração de software entre o hardware e o SO. Tal camada é chamada de Monitor de Máquina Virtual (*Virtual Machine Monitor (VMM)*), ou hipervisor, peça responsável pelo controle de quando e quais recursos físicos são entregues ao SO diretamente acima.

O hipervisor é capaz de criar, gerenciar e executar máquinas virtuais, escalonando recursos do hardware físico, *host*, para as máquinas virtuais e *guests*, que os requerem. Recursos como *Central Processing Unit (CPU)*, memória e armazenamento, são tratados pelo hipervisor como um *pool* de recursos dinamicamente alocados para os *guests* que necessitem em tempo de execução. Um hipervisor deve possuir as três propriedades listadas a seguir:

- **propriedade de equivalência**: declara que um programa em execução deve possuir comportamento idêntico seja ele executado diretamente no hardware real ou sobre virtualização, exceto por *overhead* causado pela virtualização ou escassez de recursos;
- **propriedade de eficiência**: declara que a maioria das instruções de máquinas referentes à CPU sejam executadas diretamente pelo processador físico sem a interferência do hipervisor. Esta propriedade é o maior diferencial entre hipervisor e emuladores ou interpretador;
- **propriedade de controle de recursos**: declara que o hipervisor administre todos os recursos de hardware. Não deve ser permitido que qualquer programa em uma VM acesse o hardware sem a permissão do hipervisor.

Na Figura 1 são apresentados dois tipos de VMM. O Tipo-1, que é executado diretamente sobre o host físico e controla os recursos de hardware, e tipo-2, que roda como um programa no SO do host. O acesso direto ao hardware do Tipo-1 possibilita melhor desempenho em relação ao tipo-2, que por sua vez ganha flexibilidade ao ponto de uma mesma máquina física poder executar diferentes versões do hipervisor. Porém, alguns hipervisores como o Xen possui tanto características do Tipo-1 quanto do Tipo-2 (Hwang *et al.* (2013)). Exemplos populares de hipervisores do Tipo-1 são Citrix/Xen Server, VMware ESXi e Microsoft Hyper-V. Enquanto Oracle Virtual Box e VMware Workstation são exemplo de hipervisores do Tipo-2.

Figura 1: Hipervisor **Tipo-1** vs Hipervisor **Tipo-2** .


Fonte: Adaptado de Mavridis & Karatza (2017).

Além da classificação por tipo, hipervisores podem ser classificados pelas técnicas utilizadas em sua implementação. Segundo Adams & Agesen (2006), embora a virtualização completa não demande nem a modificação do SO *guest* e nem de suporte de hardware, essas virtualizações são complexas e necessitam de técnicas avançadas para alcançar um bom desempenho, como tradução binária em tempo de execução a fim de detectar instruções e capturas instruções que não poder ser virtualizadas. A para-virtualização é a técnica que utiliza um SO modificado para chamar o hipervisor apenas quando for executar uma instrução sensível, que pode alterar o estado do sistema, evitando assim que VMM teste instrução por instrução e por consequência, atingindo desempenho superior. A virtualização assistida por *hardware*, permitida com as instruções de virtualização nos processadores modernos, pode trazer ainda mais ganhos ao possibilitar que o *guest* execute diretamente na CPU física certas instruções privilegiadas. A Tabela 1 resume as características de cada tecnologia.

Tabela 1: Comparativo entre modelos de virtualização

| | S.O. Modificado | Hardware especial | Alto Desempenho |
|---|---|---|---|
| Virtualização Completa | - | - | - |
| Para-Virtualização | ✓ | - | ✓ |
| Assistida por Hardware | - | ✓ | ✓ |

Fonte: Elaborado pelo autor (2023).

## 2.3 VIRTUALIZAÇÃO BASEADA EM CONTÊINER

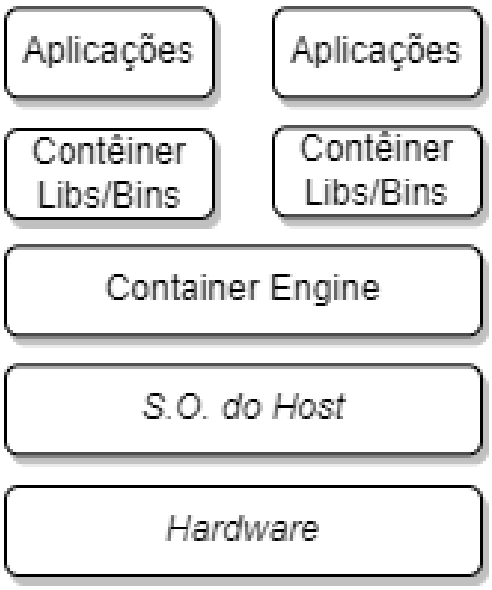
A virtualização ao nível de sistema operacional, também chamada de conteinerização, é usada para fornecer gestão e isolamento dos recursos, principalmente em ambiente Linux. Ao combinar em um mesmo pacote a aplicação, dependências, bibliotecas e arquivos de configuração, esta tecnologia possibilita uma arquitetura autossuficiente que abstrai o sistema operacional

abaixo, assim como a infraestrutura, tornando-a extremamente ágil e portátil. Segundo Soltesz *et al.* (2007), o isolamento é criado mediante três componentes principais:

- **Chroot**: permite alterar o diretório raiz de um processo e seus filhos para um local visível apenas para esta hierarquia de processos;
- **Cgroups**: permite atribuir quotas de recursos aos processos;
- **Namespaces do kernel**: permitem a todos os contêineres receberem as suas próprias configurações de rede e comunicação entre processos (IPC) e *namespaces*.

Como ilustrado na Figura 2 , este modelo permite a criação múltiplas instâncias isoladas entre si sobre o mesmo *kernel* de um único SO. Embora haja redundância, uma vez que em cada contêiner exista uma camada composta por arquivos binários e bibliotecas, BIN/LIB, a economia no compartilhamento do *kernel* do SO confere a arquitetura da Figura 2 uma substancial economia de recursos.

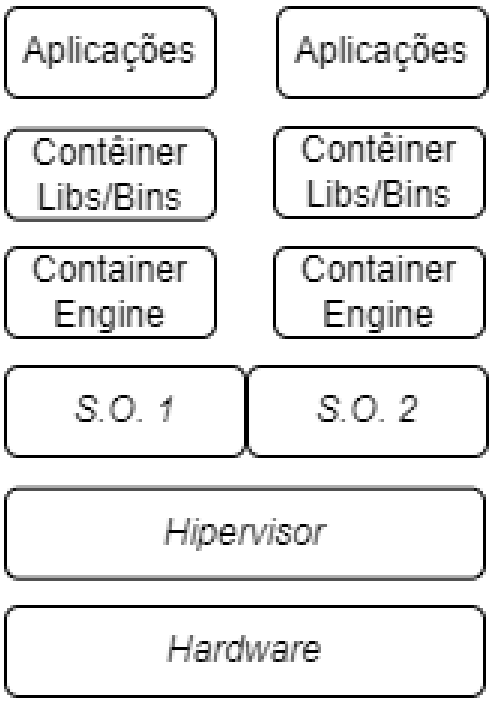
Figura 2: Contêiner em bare-metal.




Fonte: Adaptado de Mavridis & Karatza (2017)

Apesar da conteinerização descrita na Figura 2 ser eficiente ao eliminar a camada do hipervisor, redundâncias de SO, bibliotecas e binários, esta configuração não é utilizada na maior parte dos provedores de serviço em nuvem. A configuração descrita na Figura 3, na qual os contêineres são implantados em uma VM, garante ao ambiente flexibilidade na reconfiguração de recursos físicos, versionamento dos *snapshots*, tolerância a falhas, escalabilidade, etc. (al dhuraibi *et al.* (2017)). De maneira geral, a decisão é entre o desempenho de se utilizar contêineres diretamente sobre o SO do *host* ou as facilidades de se gerenciar um ambiente virtualizado com hipervisores.

Figura 3: Contêiner sobre máquina virtual.




Fonte: Adaptado de Mavridis & Karatza (2017)

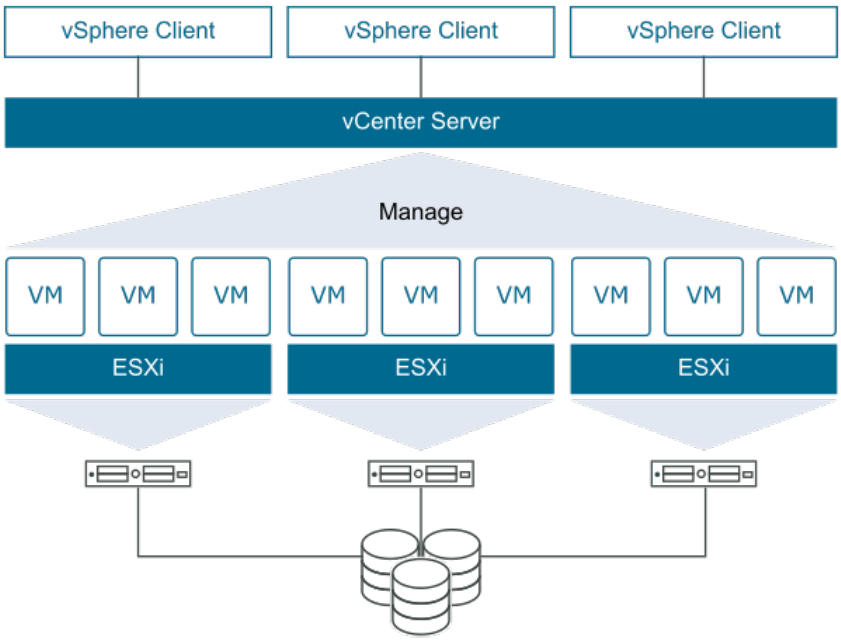
## 2.4 VMWARE VSPHERE

*VMware vSphere* VMware (2023) é uma plataforma de virtualização de servidores proprietária capaz de fornecer uma infraestrutura de virtualização completa que possibilita aos usuários criar, gerenciar e executar máquinas virtuais. A solução é composta por vários componentes, dentre os quais se destacam:

- **vCenter Server**: é o software central do ambiente. Ele gerencia, monitora e orquestra as ações sobre a infraestrutura virtual e física;
- **vSphere Web Client**: é a interface *web* que permite aos usuários acessar e gerenciar a infraestrutura vSphere via navegador *web*;
- **vSphere ESXi hypervisor**: é o hipervisor tipo-1 responsável pela virtualização.

O vSphere oferece vários recursos avançados, como alta disponibilidade, balanceamento de carga, recuperação de desastres, migração de máquinas virtuais em tempo real, *backup* e replicação de dados. Esses recursos são projetados para maximizar a disponibilidade, a segurança e a eficiência da infraestrutura de virtualização de um data center. A Figura 4 ilustra um ambiente que emprega a solução VMware vSphere, onde os usuários acessam o vCenter Server por meio do cliente vSphere. Esse servidor é responsável por gerenciar os hosts e as máquinas virtuais (VMs) usando o hipervisor ESX.

Figura 4: Estrutura do ambiente virtualizado VMware vSphere.




Fonte: VMware (2023)

## 2.5 CENTOS

O CentOS *(Community Enterprise Operating System)* é uma distribuição Linux de código aberto, baseada no código-fonte do Red Hat Enterprise Linux (RHEL), mantida pela comunidade e distribuída pela Red Hat. Segundo o levantamento *'2022 State of Open Source Report'* realizado pela OpenLogic (2022), o CentOS ainda figura como a terceira distribuição mais utilizada em servidores linux, apesar do anúncio do fim do seu desenvolvimento em dezembro de 2020. Esta fidelidade se dá pelos aspectos que tornaram a distribuição tão popular durante sua vida: estabilidade, segurança e confiabilidade. Todas as VM mencionadas neste trabalho utilizam o CentOS 7 como SO.

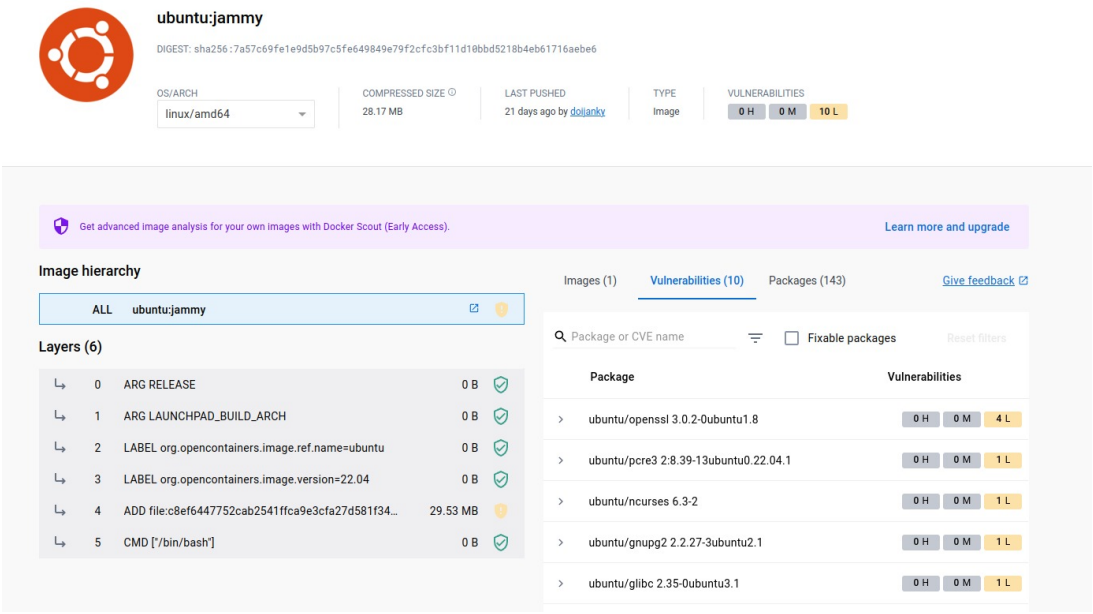
## 2.6 DOCKER

Docker (2023a) é uma plataforma de código aberto que simplifica a criação, implantação e execução de aplicativos em contêineres, que são unidades de software encapsuladas com todas as suas dependências, de código a bibliotecas, tudo necessário para a execução da aplicação. Docker utiliza imagens para rodar novos contêineres, cada contêiner é então uma instância de uma imagem em particular.

As imagens são camadas do sistema de arquivos, onde cada camada é um arquivo imutável e representa a execução de uma instrução ou conjuntos de instruções executadas na construção da imagem de contêiner. Na Figura 5, temos um exemplo das camadas utilizadas na construção de uma imagem do Ubuntu Jammy.

Além de possuírem gerenciamento simplificado, ciclo de desenvolvimento mais curto, o uso de contêineres, segundo Enberg *et al.* (2016), gera menos *overhead* de virtualização.

Figura 5: Camadas em detalhes ubuntu:jammy.




FONTE: Docker (2023b)

## 2.7 APACHE

Em abril de 2023, a W3Techs (2023) lista o Apache HTTP Server como o segundo servidor *web* mais utilizado, hospedando cerca de 32,2% dos sites conhecidos. Lançado inicialmente em 1995, é um software gratuito e de código aberto desenvolvido e mantido pela *Apache Software Foundation*. É tido pela comunidade como bastante personalizável, seguro, escalável e oferece suporte a várias tecnologias de aplicativos web. O Apache HTTP Server é o servidor *web* utilizado pela STI no sistema do SIG@.

## 2.8 GOOGLE ANALYTICS

Ferramenta de análise de dados oferecida pela *Google* que permite aos administradores de *sites* rastrear e analisar o tráfego em seus sites. O Google Analytics (2023) facilita o monitoramento de métricas importantes, como o número de visitantes, de onde eles vêm, como navegam pelo site, quanto tempo passam nele, dentre várias outras. Este serviço é utilizado pela STI e os dados capturados na matrícula do segundo semestre letivo de 2022 foi utilizado como base no dimensionamento da carga de testes descrita no Capítulo 3.

## 2.9 APACHE JMETER

Apache Jmeter (2023) é um *software* gratuito e de código aberto utilizado nos testes de aplicações e serviços *web*. Criado pela *Apache Software Foundation* em 2007, ele pode realizar testes de carga, no qual é buscado o limite de usuários simultâneos, e de stress, no qual o objetivo é determinar a capacidade de recuperação e estabilidade do sistema. O JMeter é utilizado na execução do teste de carga deste trabalho.

## 2.10 BLAZEMETER

BlazeMeter (2023) é uma plataforma *online* capaz de realizar teste de desempenho úteis na análise de desempenho de aplicações, APIs e *sites* em escala global. A plataforma oferece de forma gratuita a funcionalidade de gravação de tráfego *http* e geração roteiros no formato *JSON*, que pode ser utilizado no Apache Jmeter. O BlazeMeter foi utilizado na captura do comportamento de navegação esperado de um aluno durante a matrícula via sistema SIG@.

## 2.11 DSTAT

O Dstat (2023) é uma ferramenta leve e de código aberto utilizada para monitorar o desempenho do SO em tempo real. Capaz de oferecer dados na saída do console ou em arquivos, o Dstat consegue capturar razoáveis volumes de dados com impacto relativamente baixo no sistema. O Dstat foi utilizado na captura de consumo de recursos computacionais à nível de SO durante os experimentos deste trabalho com seguinte comando:

```
$ dstat -tcldrmsny
```

## 2.12 ZABBIX

Criado por Alexei Vladishev em 1998, o Zabbix (2023) é um software de código aberto utilizado no monitoramento de redes, servidores, máquinas virtuais e aplicações. Ele consegue monitorar a saúde e integridade de ativos de TI coletando milhares de métricas por intermédio de um agente localizado no item estudado. Possui também mecanismos de envio de alerta quando algum evento ocorre e pode também atuar de forma automatizada na resolução do problema, por exemplo, reiniciando um serviço que parou de responder. Neste trabalho o Zabbix foi utilizado na coleta de dados dos recursos computacionais utilizados pelo hipervisor e VMs.

## 2.13 ORACLE DB

O Oracle (2023) Database é um Sistema Gerenciador de Bancos de Dados Relacional (SGBDR) desenvolvido pela Oracle Corporation. É um dos bancos de dados mais utilizados por

grandes empresas por todo o mundo, conhecido por ser confiável, escalável e ter alto desempenho. O Oracle DB 11g, lançado em 2007, é o banco de dados utilizado pelo SIG@ durante a confecção deste trabalho.

# 3

# ARQUITETURA DOS AMBIENTES

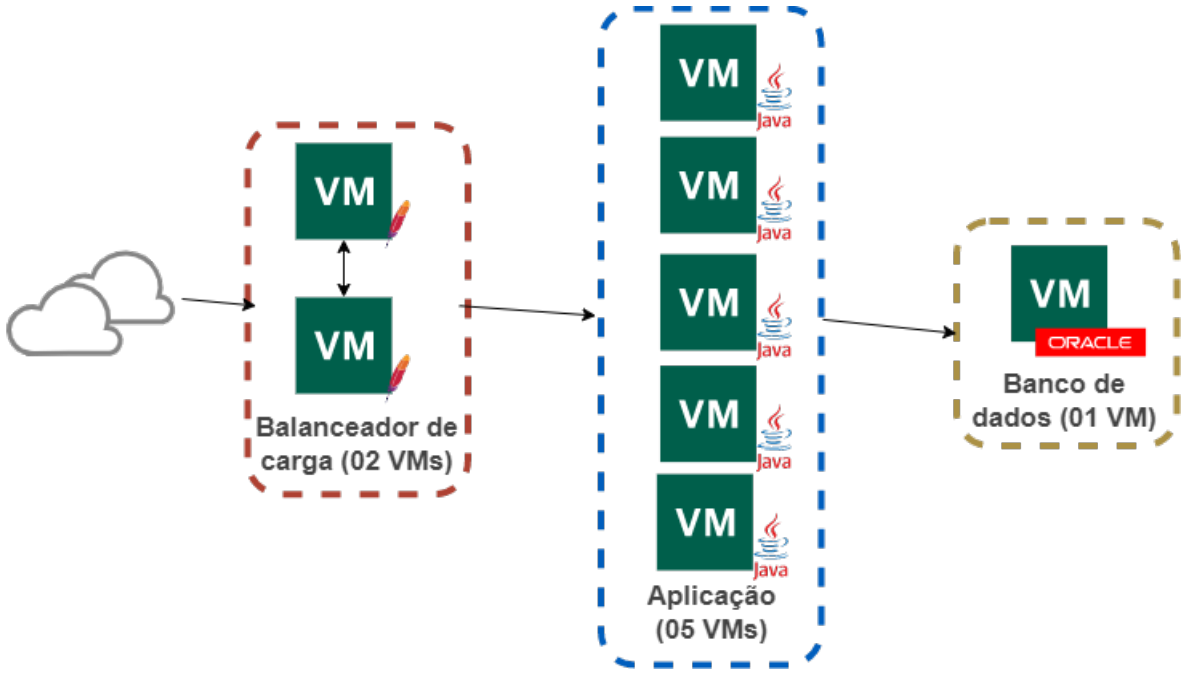
Neste capítulo serão descritos os ambientes de produção e os ambientes experimentais utilizados neste trabalho, os recursos de hardware e software utilizados.

Existem algumas arquiteturas possíveis quando se utiliza contêineres, como já mencionado na Seção 2.3. Tais arquiteturas podem variar conforme a funcionalidade que se deseja, mas sempre há o *trade-off* entre desempenho e segurança. Uma determinada arquitetura pode favorecer o alto desempenho das aplicações ou focar na segurança e facilidade no gerenciamento do ambiente, mas não é possível atingir um ótimo nas duas frentes. Segundo Mavridis & Karatza (2017), quando uma aplicação em contêiner é executada sobre máquinas virtuais, invariavelmente, haverá perdas no desempenho que podem ou não ser compensadas pelas funcionalidades de gerenciamento e segurança trazidas com a virtualização. Visando definir de quanto é este custo de desempenho quando a aplicação é conteinerizada, este trabalho cria ambientes de testes e analisa os dados coletados.

## 3.1 AMBIENTE EM PRODUÇÃO

Buscando robustez e resiliência, a arquitetura do SIG@ vem sendo aperfeiçoada com o passar dos anos. A medida que problemas eram catalogados, contramedidas foram tomadas buscando mitigá-los. Assim foi a adição de um segundo balanceador de carga, a pulverização da camada de aplicação em diversas VMs e o isolamento do banco de dados em um host físico próprio. Na Figura 6 é apresentada a estrutura atual do sistema, na qual são utilizadas oito VMs, sendo duas para o balanceamento de carga, cinco para aplicação e uma para o banco de dados.

Figura 6: Ambiente em produção (2023).




Fonte: adaptado de Clemente *et al.* (2022).

Todas as VMs rodam dentro do mesmo ambiente virtualizado VMware Vsphere, são gerenciadas pelo VMware Vcenter 6.7, utilizam o VMware ESXi 6.5 como hipervisor e estão hospedados em um *pool* de 18 servidores HP ProLiant DL360p, sendo quinze Gen8 e três Gen9. Os dados são consolidados em discos *Solid State Drive* (SSD) localizados na unidade de armazenamento HPE 3PAR StoreServ 8200, que recebe os dados dos hosts mediante uma conexão *Fiber Channel* de 16GB/s e um Switch San, modelo HP SN6000B.

As duas VMs que estão trabalhando como balanceadores de carga correspondem a camada do sistema responsável pela interação com o cliente. Elas recebem as requisições dos cliente e as encaminha para a camada seguinte, de aplicação. A sua principal função é identificar qual VM da camada seguinte possui mais recursos disponíveis e redireciona as requisições para ela, tal técnica é conhecida como balanceamento de carga. Ambas VMs utilizam o Apache Server 2.4.6 e possuem o CentOS 7 como SO *guest*.

Composta de cinco VMs idênticas rodando CentOS 7, a camada de aplicação é responsável por toda a lógica e processamento das informações do sistema. Cada VM utiliza o Apache Tomcat para hospedar a aplicação do SIG@ (versão 3.96.0.3), escrita em Java Enterprise Edition 6 (JEE 6) com uma arquitetura em camadas e utilizando o padrão Model-View-Controller (MVC) no *front-end*.

A última camada, banco de dados, possui apenas uma VM de grande porte. Responsável pelo armazenamento de dados, guarda todas as informações persistentes do sistema em um banco de dados Oracle 11g rodando sobre CentOS 7. A Tabela 2 resume o hardware virtual disponibilizado pelo hipervisor para cada VM das três camadas.

Tabela 2: Hardware virtual do SIG@ em produção

| | Balanceador-2VMs | Aplicação-5VMs | Banco-1VM |
|---|---|---|---|
| *CPU* | 4 | 4 | 12 |
| Memória | 20 GB | 20 GB | 32 GB |
| Disco | 66 GB | 55 GB | 1.46 TB |

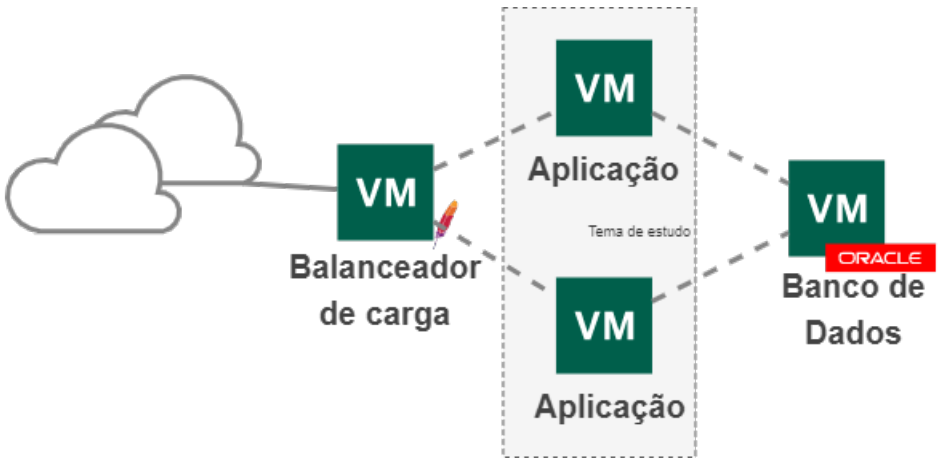
Fonte: Elaborado pelo autor (2023).

## 3.2 AMBIENTE EXPERIMENTAL

Visando responder a pergunta fundamental deste trabalho, qual o desempenho do SIG@ conteinerizado quando comparado a arquitetura atual, foram criados dois ambientes para a realização dos estudos de casos. Um representando uma versão reduzida do ambiente de produção atual e outra representando o ambiente conteinerizado, o ambiente proposto. A diferença entre elas está restrita as VMs da camada de aplicação, enquanto no cenário 1 temos uma versão reduzida do SIG@ não-conteinerizado, no cenário 2 temos a versão conteinerizada do mesmo.

A Figura 7 ilustra como estes ambientes são compostos por apenas um balanceador de carga, duas VMs na camada de aplicação e um banco de dados. Esta redução foi pensada para que todas as VMs envolvidas nos testes estivessem em um único host retirado do *pool* de servidores do DC da STI, livrando-as da concorrência com outras VMs por acesso à CPU e à memória. O host utilizado é um servidor HP ProLiant DL360 Gen 9 com 512GB de memória DDR4 2133 MHz, 2 processadores Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz de 20 núcleos cada. Este servidor está conectado a um *pool* de discos SSD na unidade de armazenamento HPE 3PAR StoreServ 7400.

Figura 7: Diagrama lógico dos ambientes de testes.




Fonte: Elaborado pelo autor (2023).

Visando reduzir o número de variáveis envolvidas e para aferir o quanto as máquinas conteinerizadas interferirão no ambiente, a máquina do balanceador de carga é a mesma utilizada nos dois cenários. O mesmo vale para a máquina de banco de dados, que deve se comportar da mesma forma independentemente da tecnologia utilizada pela aplicação. Já as VMs da camada de aplicação variam consoante o cenário do experimento. Apesar de terem o mesmo hardware virtualizado (4 vCPUs, 20 GB de *RAM* e disco com 55 GB), possuem distintas tecnologias na implantação da aplicação.

Fazendo-se uso das funcionalidades de um ambiente virtualizado, foi possível clonar algumas VMs do ambiente de produção e realizar apenas a configuração de parâmetros de configuração de rede, geração de chaves de segurança, criação de regras de firewall e outros ajustes.

O balanceador de carga utilizado nos experimentos é um clone de um dos balanceadores do ambiente de produção, porém configurado para encaminhar as requisições recebidas baseada no URL. Caso o pacote seja endereçado para sigatcc.ufpe.br, o *Apache Server* o direciona para rota '*App*', composta pelas duas VMs da camada de aplicação do cenário 1, que aqui serão denominadas App1 e App2. Caso a requisição tenha siga-docker.ufpe.br como endereço, o *Apache Server* o direciona para rota '*Contêiner*', que tem como destino as duas VMs de aplicação do cenário 2, que aqui serão denominadas Cont1 e Cont2.

Para o banco de dados foi feito um clone da VM da máquina atualmente no ambiente de produção. Para a realização das matrículas são executadas rotinas preparatórias que consistem em operações cuja finalidade é garantir a matrícula correta de cada aluno da universidade. Inicialmente, verifica-se a situação acadêmica de cada indivíduo, por exemplo, a quantidade de aprovações, cadeiras já cursadas, matérias dispensadas e perfil curricular ativo. Durante essas verificações, se aplicam regras pré-definidas que fazem sugestões personalizadas para cada aluno realizar a matrícula para o semestre seguinte. Essas sugestões ainda consideram as disciplinas ofertadas pela coordenação para o período, o número de vagas e a prioridade de cada aluno para conseguir realizar a matrícula.

As VMs da camada de aplicação do cenário 1 são idênticas às máquinas em produção, salvos alterações nas configurações de rede e chaves de acesso. Por outro lado, as VMs da camada de aplicação do cenário 2 foram construídas do zero, mantendo-se apenas a versão do SO e hardware disponibilizado. Nestas VMs foi instalado o Docker Engine - Community na versão 20 para executar a imagem do SIG@ conteinerizado desenvolvida pela equipe de desenvolvimento da STI. A Tabela 3 resume as características das VMs criadas para os casos de estudos.

Tabela 3: VMs utilizadas nos experimentos.

| **Experimentos** | Balanceador | App1, App2 | Cont1, Cont2 | Banco |
|---|---|---|---|---|
| Aplicação | Apache Server | Siga não-conteinerizado | Siga Docker | Oracle DB 11g |
| *CPU* | 4 | 4 | 4 | 12 |
| Memória | 20 GB | 20 GB | 20 GB | 32 GB |
| Disco | 66 GB | 55 GB | 55 GB | 1.46 TB |
| Cenário 1 | ✓ | ✓ | X | ✓ |
| Cenário 2 | ✓ | X | ✓ | ✓ |

Fonte: Elaborado pelo autor (2023).

O processo da criação da imagem Docker foi iniciado com uma análise de quais softwares participam do sistema do SIG@ e como eles se relacionam. Partindo deste levantamento, foi possível construir o *Dockerfile*, arquivo que contém uma série de instruções que definem a configuração do ambiente em que o aplicativo será executado, incluindo a escolha do sistema operacional, a instalação de bibliotecas e dependências, a cópia de arquivos de código-fonte e a configuração de variáveis de ambiente. Em seu conteúdo, o *Dockerfile* do SIG@ define as variáveis de ambiente sobre uma imagem base do SO, CentOS 7, instala a versão apropriada do Java com suas dependências, assim como o Tomcat, permissões de usuário e configura os acessos de rede. Por razões de segurança, informações mais detalhadas sobre estas configurações não podem ser divulgadas. O limite máximo de memória a ser utilizado pela aplicação em ambos os cenários é definido pelas configurações da Java Virtual Machine (JVM), listadas na Tabela 4.

### 3.2.1 SIG@ Tradicional

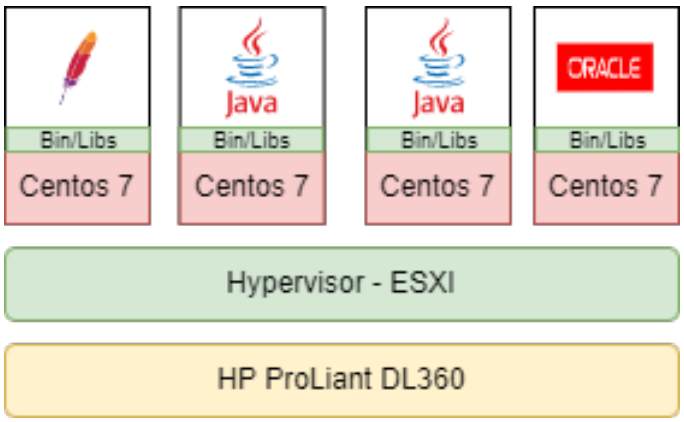
Este ambiente é uma versão simplificada do SIG@ atualmente em produção. Em vez de oito máquinas virtuais, foram utilizas quatro. Foi retirado o segundo balanceador de carga, que tem como principal função evitar um ponto único de falha no sistema, reduziu-se de cinco para duas as máquinas virtuais com as aplicações SIG@ e manteve-se um banco de dados Oracle 11, como apresentado na Figura 8.

Tabela 4: Configuração de Memória da Java Virtual Machine para a execução do SIG@.

| **Parâmetro** | **Valor** |
|---|---|
| Heap Size | 15 GB |
| Max Heap Size | 15 GB |
| Permgen Size | 512 MB |
| Max Permgen Size | 512 MB |

Fonte: Elaborado pelo autor (2023)

Figura 8: Infraestrutura do cenário análogo ao de produção.

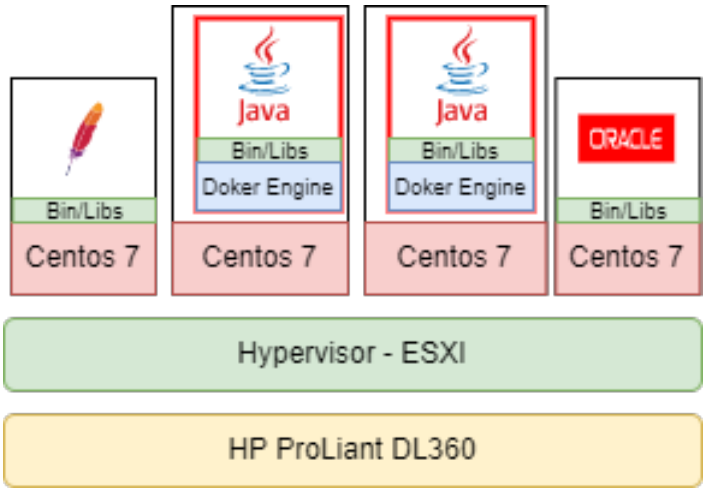



Fonte: Elaborado pelo autor (2023).

### 3.2.2 SIG@ Conteinerizado

Neste ambiente, o balanceador é mantido, assim como o banco de dados. A alteração se dá nas máquinas virtuais que abrigam a aplicação. Estas duas máquinas virtuais utilizam a distribuição Linux CentOS 7 e utilizam a Docker Engine - Community na versão 20 para executar a imagem do SIG@ Docker. Figura 9 ilustra o ambiente de teste na versão conteinerizada.

Figura 9: Infraestrutura do cenário conteinerizado.




Fonte: Elaborado pelo autor (2023).

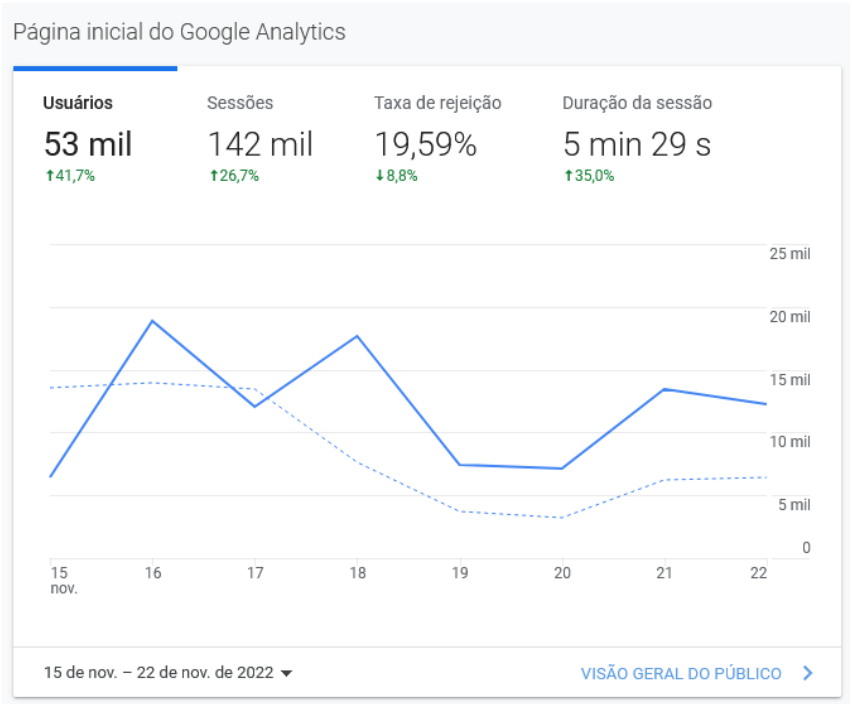
## 3.3 CARGA DE TESTES

Em um período de matrículas típico, que dura por volta de uma semana, o SIG@ da UFPE atende cerca de 30 mil alunos ativos. A fim de evitar a sobrecarga do sistema, o acesso é liberado apenas para determinados centros acadêmicos conforme uma escala pré-divulgada pelos meios oficiais da instituição. Desta forma, consegue-se controlar os picos de acessos simultâneos

ao sistema e o volume de escrita no banco de dados. Como exemplo, na Figura 10 é apresentado o somatório de acessos diários realizados ao SIG@ durante o período de matrícula do segundo semestre letivo de 2022. Tal informação foi obtida através do Google Analytics, onde é possível observar que o pico de acessos foi atingido no dia 16 de novembro, com 7034 matrículas ao longo de todo o dia. Além disso, houve por volta de 2800 acessos entre as dez e as onze horas do mesmo dia.

Todavia, estas requisições não representam a totalidade dos acessos ao sistema, uma vez que através do *Google Analytics* foram registrados cerca de 2,8 mil usuários no mesmo intervalo de uma hora. Como indicado na Figura 11, durante a semana de matrículas foram contabilizados 141.965 sessões de 53.807 usuários, totalizando 425.112 visualizações de páginas. Neste período, um usuário acessa o sistema em média 2,69 vezes e a cada acesso visualiza, em média, 2,99 páginas num período de 5 minutos e 30 segundos, sendo que em 19,59% das vezes ele não entra no sistema, apena carrega a tela inicial.

Figura 11: Quantidade de acessos ao SIG@ durante o período de matrículas para o semestre 2022.2.

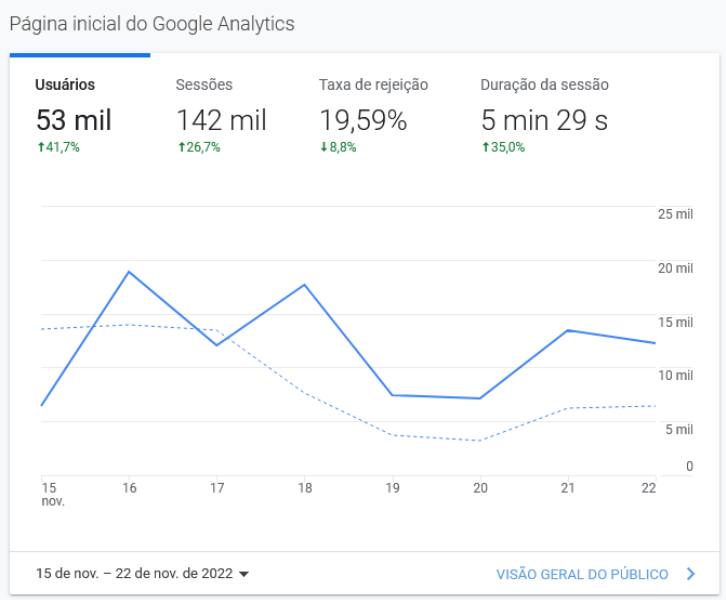



Fonte: Captura de tela realizada pelo autor da ferramenta Google Analytics (2023).

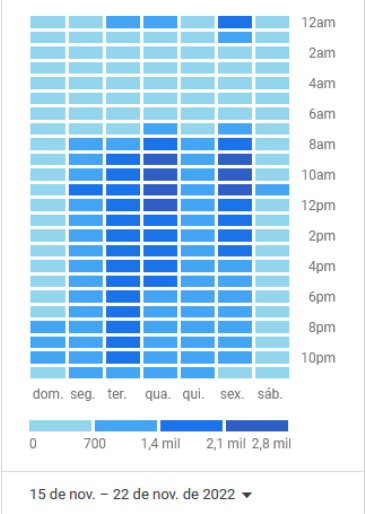
Outro dado interessante capturado pelo *Google Analytics* é sobre a relação do tipo de dispositivo utilizado pelo usuário na navegação do sistema com o seu intuito do acesso. Conforme ilustrado na Tabela 5, o telefone celular é o dispositivo utilizado em mais da metade dos acessos ao SIG@, porém para confirmar a matrícula nas disciplinas do semestre letivo, ele não é empregado nem por 30% dos usuários, ao passo que o computador é utilizado por 69% com a finalidade de realizar a matrícula. Isto indica que geralmente o aluno da UFPE realiza de um a dois acessos prévios ao sistema, preferencialmente no telefone celular, antes de confirmar a

Figura 10: Acessos ao SIG@ durante matrículas de 2022.2

(a) Acessos por dia




(b) Acessos por hora




Fonte: Captura de tela realizada pelo autor da ferramenta Google Analytics (2023).

Tabela 5: Dispositivos utilizados no acesso ao SIG@ e no momento de confirmar a matrícula.

| | Acessos | Matrícula |
|---|---|---|
| Computador | 48,9% | 69,0% |
| Celular | 50,1% | 29,8% |
| *Tablet* | 1,0% | 1,2% |

Fonte: Elaborado pelo autor (2023).

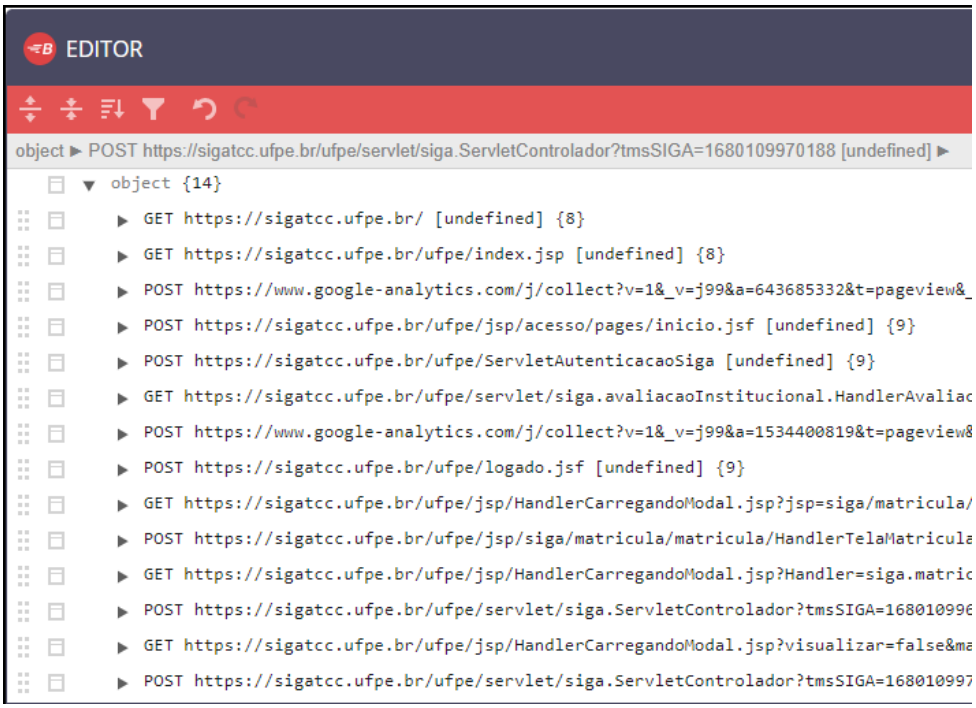
matrícula utilizando um computador.

Quando o tráfego capturado durante o período é analisado sob a ótica da estrutura física do sistema (Figura 6), é possível afirmar que cada servidor da camada de balanceamento de carga tratou no máximo 1400 usuários por hora e que cada servidor da camada de aplicação recebeu não mais que 560 usuários no mesmo intervalo. Apesar de valiosos, estes dados não possuem granularidade suficiente para determinarmos a quantidade de sessões simultâneas atendidas pelo ambiente neste intervalo. Porém, se assumirmos o pior caso, no qual cada usuário gera duas 2 sessões dentro do intervalo de uma hora, podemos determinar a carga máxima aplicada no sistema. Assim, cada VM da camada de aplicação trata 1120 sessões neste intervalo com 102,67 acessos simultâneos. Esse valor de acessos simultâneos pode ser calculado utilizando-se a Equação 3.1, onde *NUC* é o número de usuários concorrentes, *NSH* é o número de sessões por hora, *DMS* é a duração média da sessão em segundos e *TU* é tempo de utilização em horas.

$$NUC = \frac{\frac{NSHxDMS}{3600}}{TU} \tag{3.1}$$

Definida a carga de trabalho que será aplicada sobre o objeto de estudo, as máquinas virtuais da camada de aplicação, um *plugin* do *Blazemeter* para navegador foi utilizado na captura do comportamento simulado de um usuário no ato da matrícula. Uma vez que o comportamento de navegação interna ao sistema não é monitorado pelo Google Analytics (2023) por razões de privacidade e segurança, foi criado um fluxo de cliques visando gerar o impacto similar aos dos acessos discutidos anteriormente, como exposto na Figura 12.

Figura 12: BlazeMeter, ferramenta utilizada na captura das ações para matrícula.




Fonte: Captura de tela realizada pelo autor da ferramenta BlazeMeter (2023) .

Esta série de requisições do protocolo Hypertext Transfer Protocol (HTTP) indicam a ordem e conteúdo das mensagens trocadas entre a aplicação do SIG@ e o navegador do usuário. Estas informações são utilizadas como base do *script* da aplicação que será utilizada na ferramenta responsável pelo teste de carga, o *Apache Jmeter*. Com o uso de variáveis, parametrizações e condições para tratamento de erros, chega-se a um roteiro de ações que pode ser executado de forma controlada na simulação dos acessos. O algoritmo 1 ilustra as atividades do *script* codificado. Ele carrega uma lista de CPFs previamente selecionados ( 6 mil), alunos aptos a realizar a matrícula com ao menos uma sugestão automática de disciplina fornecida pelo sistema, e os utiliza como base durante a execução dos testes.

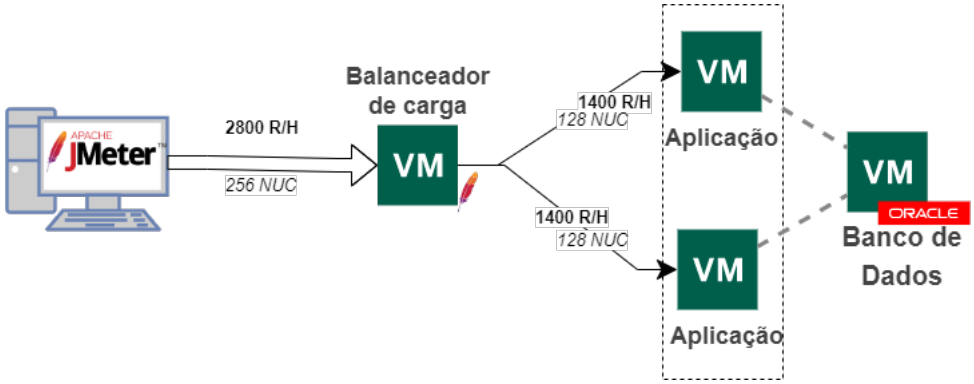
Para replicar a carga de trabalho observada durante o pico de acessos para o período de matrícula em 2022.2, no qual foi registado 1120 sessões no período de uma hora e calculado 103 sessões simultâneas por VM na camada de aplicação, teríamos de executar o *script* para 2247 alunos durante a duração do experimento, uma hora. Porém, foi utilizada uma carga 25% maior, 2800 acessos, dando a este teste características de um teste de carga, uma vez que o sistema

**Algoritmo 1:** Pseudocódigo do Script Jmeter

```
1  for Aluno ← CPF.txt to EOF do
2      Abre página inicial do SIG@
3      Envia CPF e senha do Aluno
4      Recebe Hash de acesso
5      Carrrega página de logado com sucesso
6      Carrega módulo de matrícula
7      Aceita sugestão de uma cadeiras à ser cursada
8      Confirma a matrícula
9      Fecha janela de popup de impressão de comprovante
10     Desloga do sistema
11 Gera relatório
```

recebe um volume de acesso semelhante à carga de pico durante um período relativamente longo a e assim obter métricas da qualidade do acesso e o uso de recursos (de Sousa Santos & dos Santos Neto (2008)). Esta quantidade de acessos, quando distribuídos no intervalo de uma hora, fornecem 128 acessos simultâneos a cada VM da camada de aplicação, conforme indicando na Figura 13.

Figura 13: Carga de testes executada pelo *Apache Jmeter* sobre o sistema.




Fonte: Elaborado pelo autor (2023).

Durante a execução dos testes os dados sobre o uso de recursos computacionais foram capturados à nível de SO pelo *Dstat* e ao nível de hipervisor pelo *VCenter*.

# 4

# RESULTADO DOS EXPERIMENTOS

Para cada teste em ambos os cenários propostos, o *script* discutido na Sessão 3.3 é executado pelo *Apache Jmeter* em um computador externo a infraestrutura estudada durante o período de uma hora. Neste período são enviadas 2800 requisições de matrícula separadas entre si pelo intervalo de 0,77 segundos.

Durante os testes, cada cenário recebe sua carga de requisições, o *Dstat* é executado em cada uma das VMs com parâmetros que o faz coletar uma amostra a cada segundo durante setenta minutos, totalizando 4200 medições por VM. Enquanto isto, o hipervisor envia suas métricas de utilização de recursos ao sistema de gerenciamento do Vcenter, que os repassa ao Zabbix, onde são guardadas e visualizadas. No decorrer de cada teste, as VMs não pertencentes ao cenário são desligadas visando não consumir recursos e influenciar as medições que estão sendo realizadas. No **cenário 1** a carga de trabalho é direcionada para o SIG@ não conteinerizado. Já no **cenário 2** as requisições têm como destino a versão conteinerizada do sistema.

Cada cenário recebeu cinco testes de carga e responderam de forma similar às requisições. Enquanto o cenário 1 realizou em média 2675,2 das 2800 matrículas demandas com o tempo médio de 2215 ms por requisição, o cenário 2 atendeu 2676,4 matrículas em média no com o tempo médio de resposta de 2198 ms.

Visando apresentar o volume de dados, os resultados foram exibidos nas subseções seguintes através de diagramas de caixa. Esses diagramas agregam as medições de cada máquina virtual juntamente com as VMs que compõem seus respectivos cenários, facilitando a visualização do uso de recursos no ambiente de teste. Foram selecionadas quatro métricas primárias para essa visualização, sendo uma referente ao uso de *CPU*, outra ao uso de disco, uma terceira ao uso de rede e a última ao uso de memória.

Considerando que todas as VMs testadas estavam hospedadas em um servidor ProLiant DL360 Gen9 com 2 processadores, cada um com 10 núcleos e 20 threads, o uso percentual de CPU representa o valor máximo de processamento fornecido à VM pelo hipervisor, dependendo da quantidade de processadores lógicos configurados. As demais métricas se referem a valores absolutos. O uso de memória é medido em megabytes de *RAM* ocupados. Já o uso de rede é expresso em megabytes por segundo do fluxo de rede agregado, ou seja, a soma do que a VM recebe e envia por segundo através de sua interface de rede. Por fim, o uso de disco é medido em

IOPS, que em português significa "operações de entrada/saída por segundo".

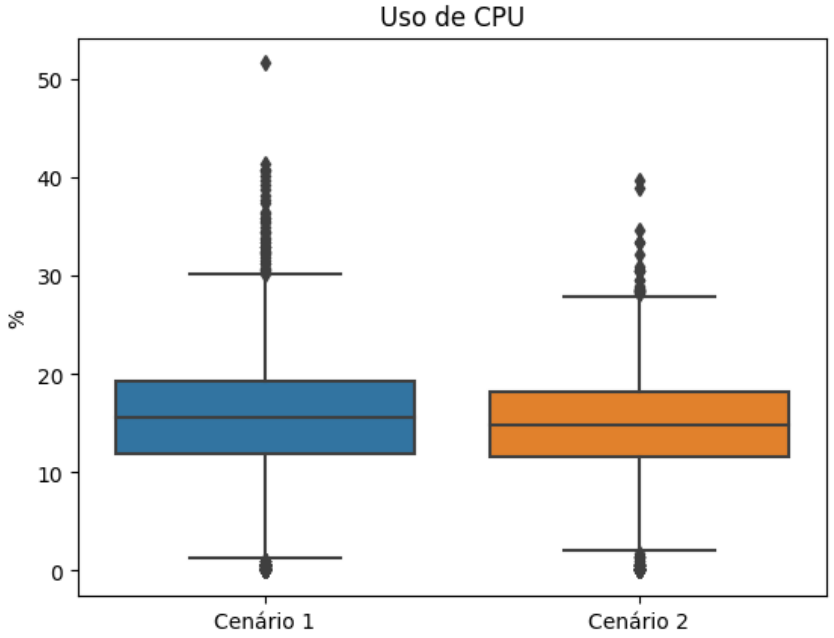
## 4.1 BALANCEADOR DE CARGA E BANCO DE DADOS

Tanto na camada de balanceamento de carga quanto na camada de banco de dados do sistema, apenas uma máquina virtual foi utilizada nos ambientes experimentais discutidos neste trabalho. Estas duas VMs são reutilizadas em ambos os cenários sem modificações em todas as cinco repetições de cada cenário. Após cada realização de teste, cada VM tem seu estado anterior restaurado mediante uma funcionalidade do *Vcenter*, o snapshot. Ao todo elas são utilizadas dez vezes, sendo que para cada utilização são geradas 4200 amostras de dados, totalizando 42 mil.

### 4.1.1 Balanceador de Carga

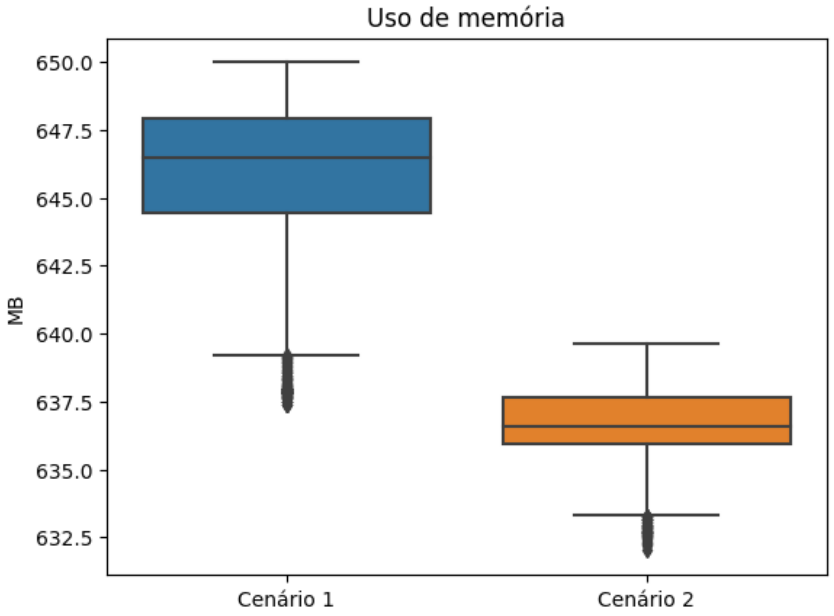
Responsável por receber as requisições dos usuários por meio de Apache Server 2.4.6, esta VM possui o SO CentOS 7, 20 GB de memória RAM e 4 vCPUs. Historicamente, apresenta baixas taxas de utilização de disco, memória e CPU, porém valores altos na utilização de rede.

Figura 14: Processamento do Balanceador de Carga.



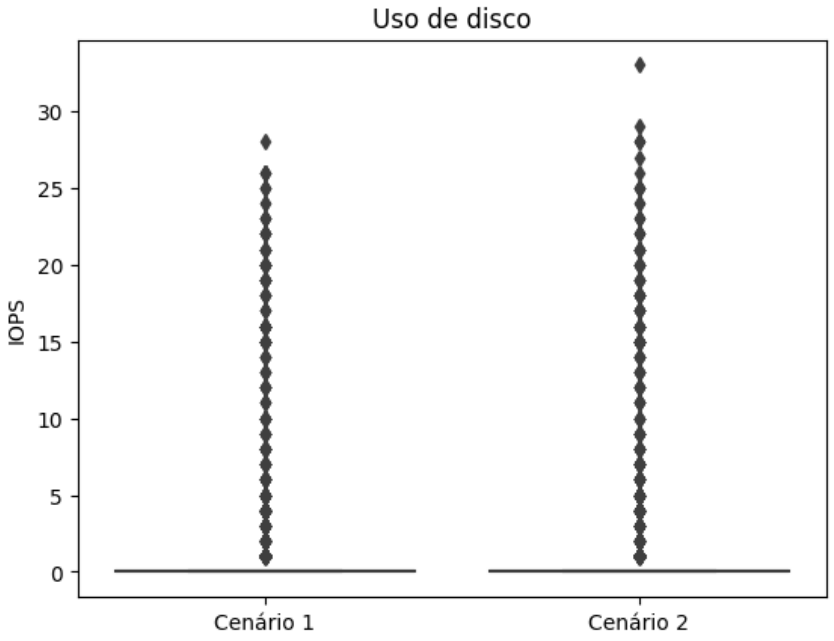

Fonte: Elaborado pelo autor (2023).

Figura 15: Utilização de Memória RAM pelo Balanceador de Carga.



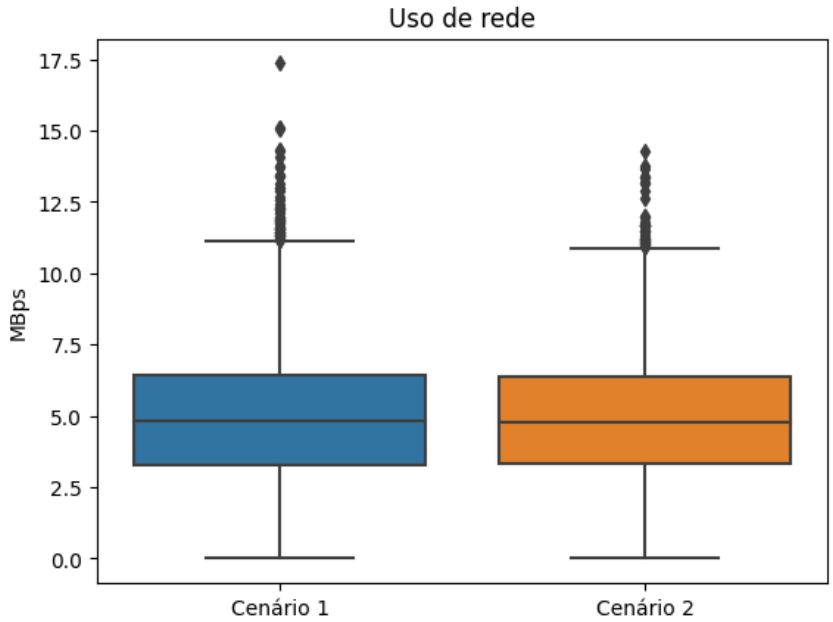

Fonte: Elaborado pelo autor (2023).

Figura 16: Instruções de entrada e saída por segundo no Balanceador de Carga.




Fonte: Elaborado pelo autor (2023).

Figura 17: Tráfego de rede no Balanceador de Carga.



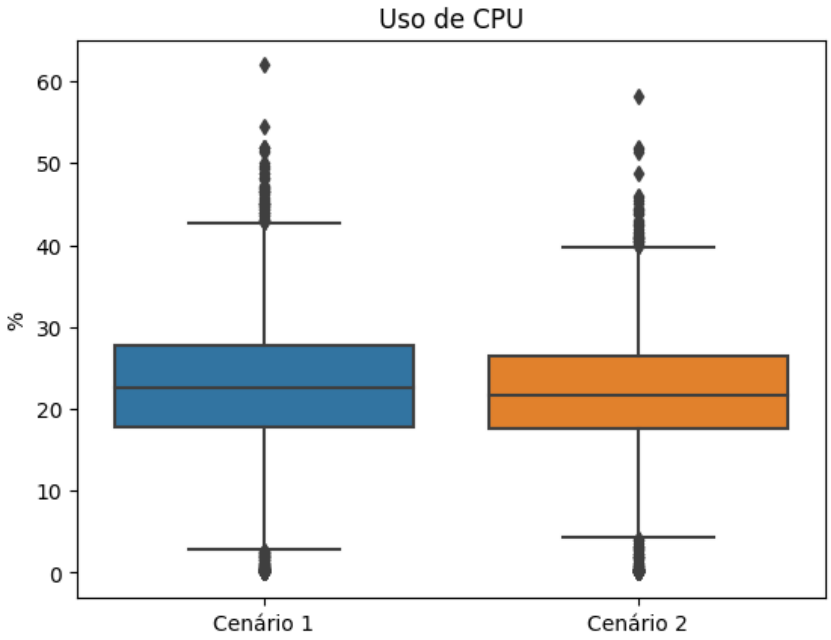

Fonte: Elaborado pelo autor (2023).

As Figuras 14, 15, 16 e 17 agrupam as 42 mil medições que descrevem o comportamento da VM de balanceamento de carga durante os dez testes realizados. Os resultados obtidos para o uso de *CPU*, escrita em disco e uso de rede apresentam uma grande similaridade, enquanto o uso de memória no cenário conteinerizado é ligeiramente inferior. No entanto, ao compararmos as medianas mostradas na Figura 15, que são de 646,4 MB no primeiro cenário e 636,6 MB no segundo, com o total de memória disponível para a máquina virtual, que é de 20 GB, percebemos que a diferença é desprezível.

### 4.1.2 Banco de Dados

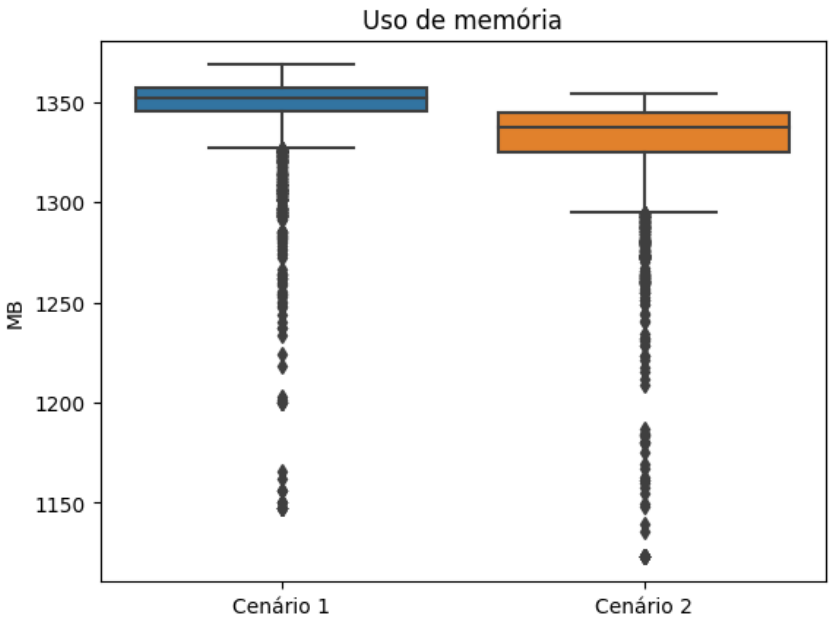
É composto por uma única VM que concentra a escrita de todas as informações pertinentes do sistema em uma instância do Oracle DB 11g que ocupa cerca de 1,5 TB. Possui 32 GB de memória RAM, 12 vCPUs e historicamente faz o uso intensivo de disco e memória.

Figura 18: Processamento do Banco de Dados.



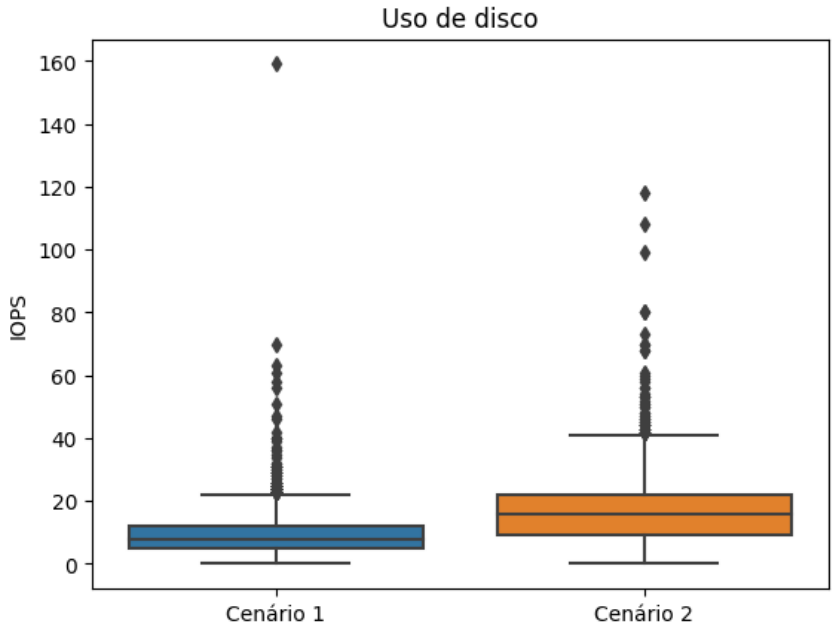

Fonte: Elaborado pelo autor (2023).

Figura 19: Utilização de Memória RAM pelo Banco de Dados.



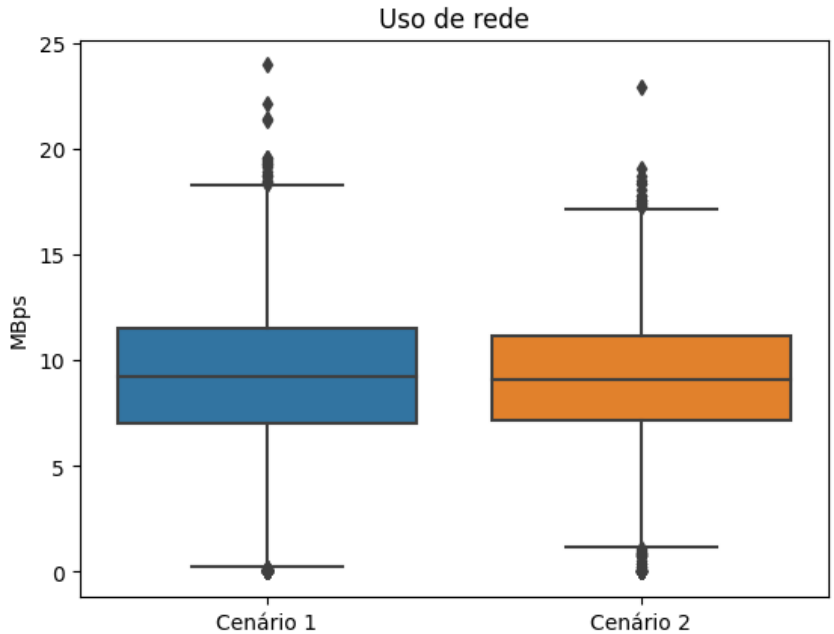

Fonte: Elaborado pelo autor (2023).

Figura 20: Instruções de entrada e saída por segundo no Banco de Dados.




Fonte: Elaborado pelo autor (2023).

Figura 21: Tráfego de rede no Banco de Dados.




Fonte: Elaborado pelo autor (2023).

As Figuras 18, 19, 20 e 21 agrupam as 42 mil medições que descrevem o comportamento da VM de banco de dados durante os dez testes realizados. Os resultados obtidos para as quatro métricas estudadas apresentam alto grau de similaridade, sendo que o uso de memória no cenário conteinerizado é ligeiramente inferior e o uso de disco é ligeiramente maior. No entanto, como

era esperado, essas diferenças não são significativas e podemos afirmar que o desempenho do banco de dados nos dois cenários é equivalente.

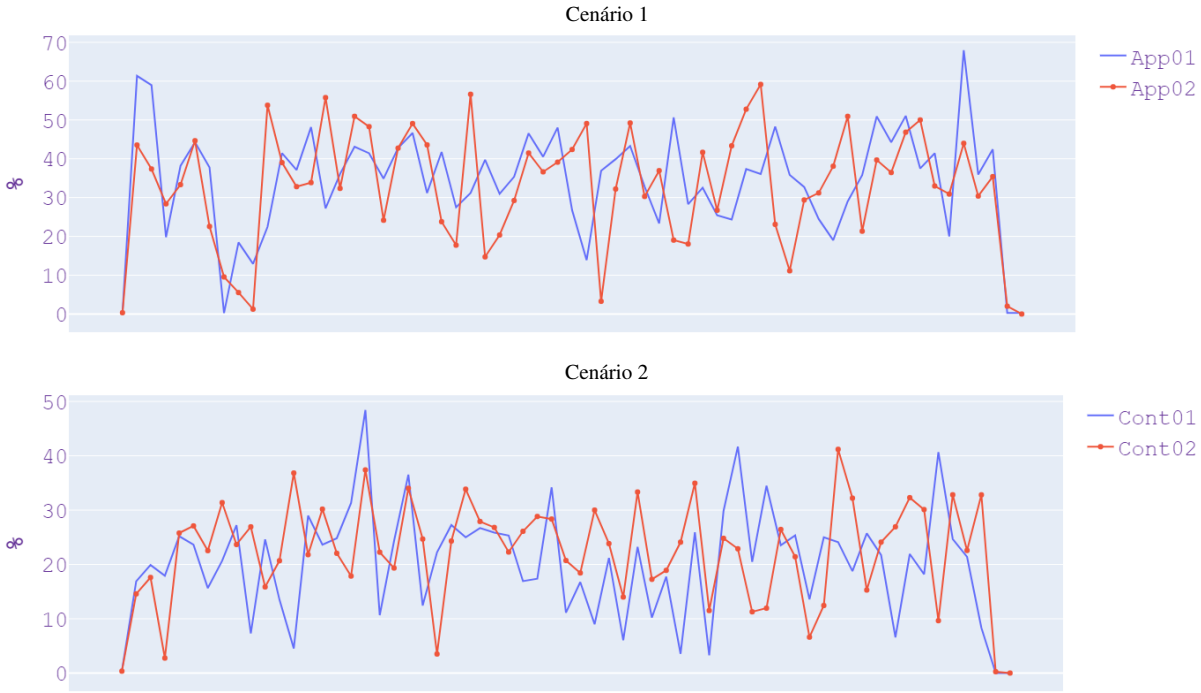
## 4.2 CAMADA DE APLICAÇÃO

Embora as duas soluções de implementação do SIG@ tenham atendido as solicitações de matrículas de forma similar, cerca de 96% das requisições de matrícula com sucesso, o consumo de recursos computacionais apresentou particularidades em cada cenário. Através das métricas apresentadas e discutidas nas seguintes subseções, o custo computacional de cada um dos cenários será exposto.

### 4.2.1 Uso de CPU

Toda tecnologia de virtualização possui sobrecargas características. Ao passo que o cenário 1 possui todos os *overheads* inerentes ao uso do hipervisor, o cenário 2 herda todas as sobrecargas conhecidas da virtualização por VM e ainda tem as penalidades de desempenho da virtualização ao nível de sistema operacional. Segundo Enberg *et al.* (2016), as sobrecargas do hipervisor ligadas ao uso de *CPU* são originadas pelo agendamento duplo, equidade de escalonamento, gestão de interrupções e assimetria nas *vCPU*.

Figura 22: Uso de *CPU* nas *VMS* da camada de aplicação vs Tempo.

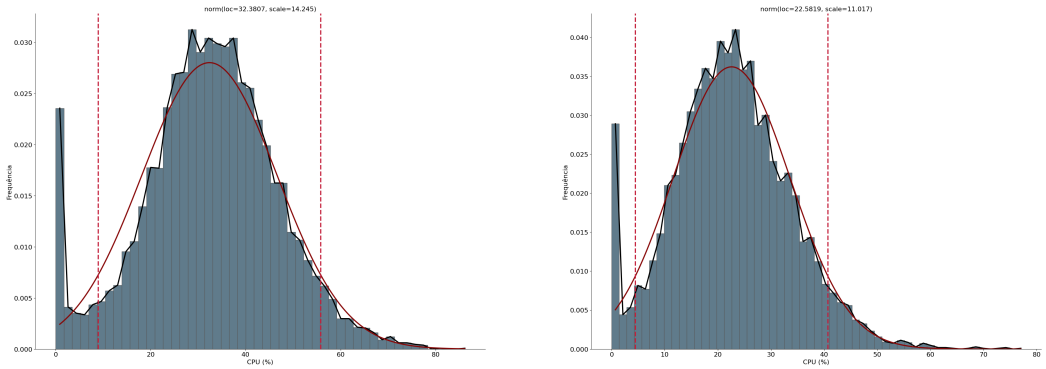



Fonte: Elaborado pelo autor (2023).

A Figura 22 mostra a utilização da capacidade total de processamento de cada VM

Figura 24: Uso agregado de *CPU* em % ajustado a curva normal

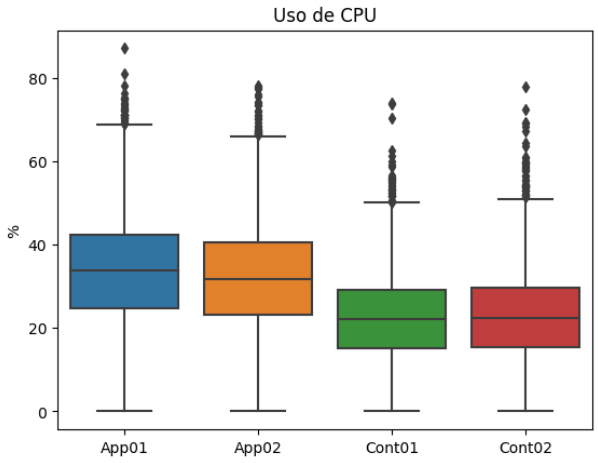
(a) App01 e App02 (b) Cont01 e Cont02




Fonte: Elaborado pelo autor (2023).

durante o primeiro teste de carga recebido pelo ambiente. Nela, nota-se uma grande variação de valores, especialmente no ambiente não conteinerizado. Tal flutuação no uso de *CPU* é comum, mas requer atenção especial para a alocação de máquinas virtuais com comportamento similar no mesmo servidor físico, uma vez que várias delas podem solicitar acesso ao processador ao mesmo tempo, sobrecarregando o escalonador do hipervisor.

Figura 23: *Boxplot* do uso de *CPU* das *VMS*.




Fonte: Elaborado pelo autor (2023).

Já a Figura 23 agrega os valores captados nos dez testes realizados, cinco em cada cenário, e os dispõem lado a lado mostrando a simetria entre as máquinas envolvidas no cenário. A Figura 24 mostra o quão a distribuição normal se ajusta aos dados observados com paramêtros $\mu$ =32.38 e $\sigma$ = 14.21 no cenário 1 e $\mu$ =22.58 e $\sigma$ = 11.02 para o cenário 2, com exceção aos picos de utilização nula da *CPU* nos dois gráficos que correspondem aos instantes finais do experimento, nos quais a carga de trabalha já havia sido processada. A Tabela 6 referendas os valores apresentados nas figuras anteriores. Nela observamos que a média se aproxima da

Tabela 6: Utilização percentual de *CPU* nas 21 mil medições.

| | **App01** | **App02** | **Cont01** | **Cont02** |
|---|---|---|---|---|
| Média | 33.35 | 31.41 | 22.38 | 22.78 |
| Desvio Padrão | 14.38 | 14.04 | 10.79 | 11.24 |
| Mínimo | 0.0 | 0.0 | 0.0 | 0.0 |
| 25% | 24.81 | 23.17 | 15.24 | 15.50 |
| 50% | 33.76 | 31.75 | 22.11 | 22.51 |
| 75% | 42.45 | 40.42 | 29.20 | 29.69 |
| Máximo | 87.02 | 78.10 | 73.89 | 77.75 |

Fonte: Elaborado pelo autor (2023)

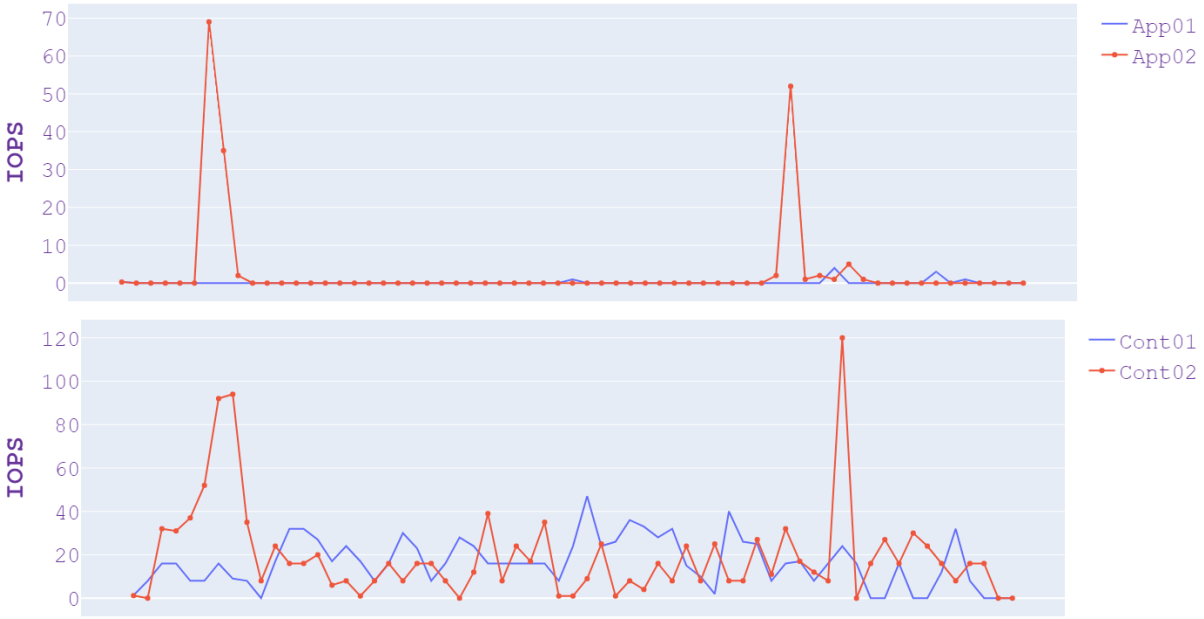
mediana, comportamento típico da distribuição normal. Outro fato importante observado é a alta variância na utilização da *CPU* em ambos cenários e o uso levemente inferior no ambiente conteinerizado.

### 4.2.2 Uso de disco

O tempo de acesso ao disco é ordens de grandeza maior que tempo de acesso à memória, logo quantos menos acesso a disco, mais rápida tende a ser a aplicação. Enquanto o hipervisor lida com *overhead* relacionado ao escalonamento das requisições de acesos ao disco vindo de múltiplas VMs, uma aplicação conteinerizada lida com o atraso relacionado a estrutura em camadas do sistema de arquivos que compõe um contêiner. A Figura 25 mostra quantas instruções de leitura ou escrita, IOPS, foram geradas durante o primeiro experimento em cada cenário. Nela é clara a discrepância de comportamento entre os dois ambientes, visto que o ambiente em contêiner gera instruções de acesso ao disco constantemente, enquanto as máquinas no ambiente não-conteinerizado raramente acessam o disco, mas quando o fazem, é em grandes volumes.

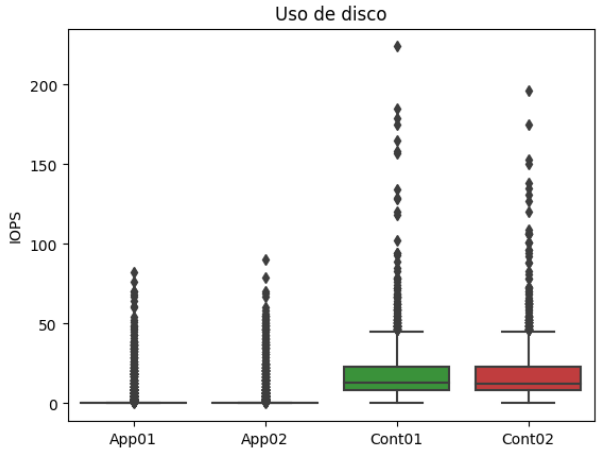
A Figura 26 agrega os valores de instruções de escrita/leitura em disco (IOPS) geradas nos dez testes realizados, cinco em cada cenário, e os dispõem lado a lado. Apesar de ambos os cenários apresentarem valores médios baixos, 2 IOPS no primeiro e 15 no segundo, graficamente podemos notar que o cenário de containers gerou mais *outliers* relevantes. Este comportamento não é desejável, pois tem poder de contribuir na formação de gargalos no acesso à unidade de armazenamento.

Figura 25: Comparativo do uso de disco *CPU* nas *VMS* da camada de aplicação.




Fonte: Elaborado pelo autor (2023).

Figura 26: *Boxplot* do uso de disco das *VMS*.




Fonte: elaborado pelo autor (2023).

Tabela 7: Utilização de discos medidos em IOPS nas 21 mil medições.

| | **App01** | **App02** | **Cont01** | **Cont02** |
|---|---|---|---|---|
| Média | 1.56 | 1.60 | 15.16 | 15.28 |
| Desvio Padrão | 6.23 | 6.73 | 15.28 | 15.03 |
| Mínimo | 0.0 | 0.0 | 0.0 | 0.0 |
| 25% | 0.0 | 0.0 | 8.0 | 8.0 |
| 50% | 0.0 | 0.0 | 13.0 | 12.0 |
| 75% | 0.0 | 0.0 | 23.0 | 23.0 |
| Máximo | 82.0 | 90 | 224.0 | 196 |

Fonte: Elaborado pelo autor (2023)

### 4.2.3 Uso de memória

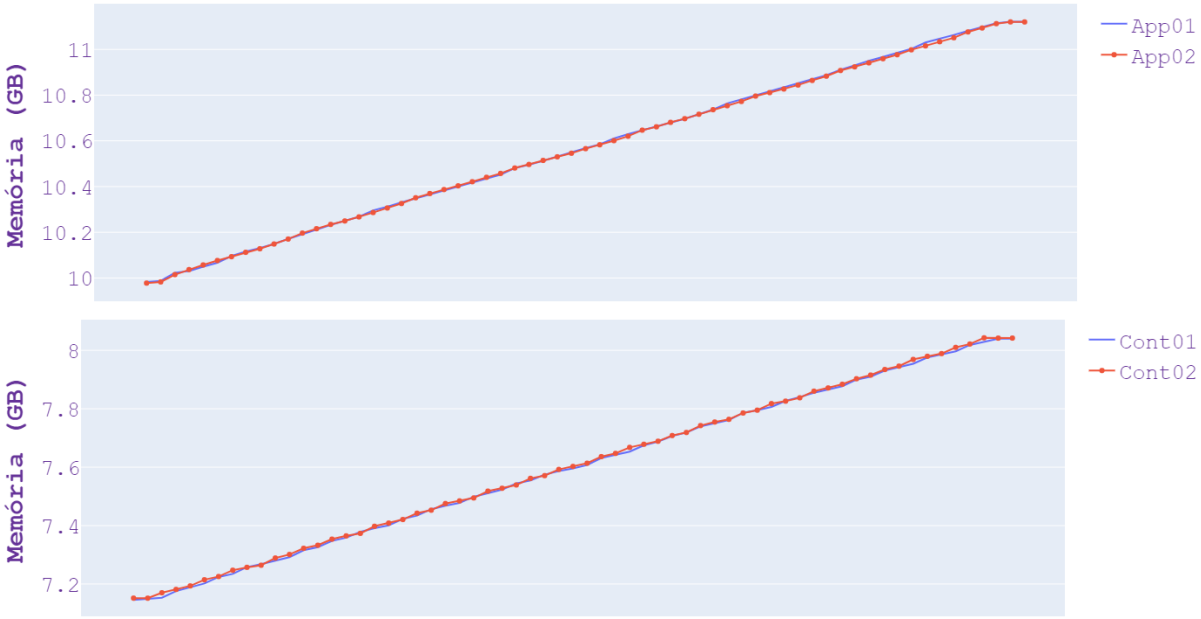
Tanto a virtualização baseada em hipervisores, quanto a virtualização ao nível de sistema operacional, gera degradação de desempenho no uso de memória.

Dois mecanismos são responsáveis por esta degradação de desempenho, perda na recuperação de memória(*ballooning*) e a deduplicação. O *Ballooning* é uma técnica utilizada pelos hipervisores modernos que consiste em gerar um processo no SO de suas VMs *guests* com a intenção de ter posse sobre parte da memória de VM e por redistribuí-lo quando necessário. Já a deduplicação é uma técnica que visa diminuir o consumo de memória ao identificar que conteúdos iguais repetidos. O hipervisor comumente utiliza a deduplicação quando possui mais de uma VM com o o mesmo SO e o *container engine* utiliza esta técnica para evitar que uma lib ou arquivo seja alocado em memória mais vezes do que o necessário Enberg *et al.* (2016). Como o cenário 2 emprega as duas tecnologias de virtualização, contêiner sobre máquinas virtuais gerenciadas por hipervisor, ele sofre duplamente com este *overhead.*

A Figura 25 mostra a utilização de memória RAM durante o primeiro experimento em cada cenário. O comportamento do gráfico em ambos cenários sugere que a aplicação tende a reter dados referente a sessão ativa na memória sem a devida liberação no período observado. Além disto, é observado que as máquinas do cenário conteinerizado no momento de início da medição consomem quase 3 GB a menos que as VMs do primeiro cenário.

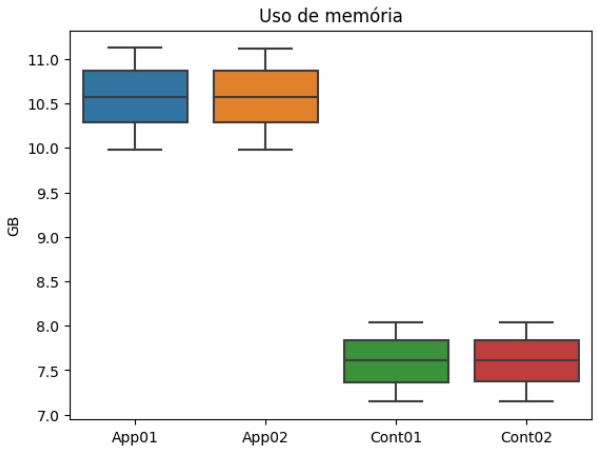
Já Figura 28 agrega os valores de ocupação de memória RAM capturadas nos dez testes executados, sendo cinco em cada cenário, e os dispõem lado a lado. A simetria deste gráfico mostra a consistência no consumo de memória durante os experimentos. Este comportamento pode ser explicado pela recuperação de *snapshot* realizado antes do início de cada teste, que força que cada máquina comece cada teste com o mesmo estado salvo.

Figura 27: Comparativo do uso de memória nas *VMS* da camada de aplicação.




Fonte: Elaborado pelo autor (2023).

Figura 28: *Boxplot* do uso de memória das *VMS*




Fonte: Elaborado pelo autor (2023).

.

Tabela 8: Utilização de memória *RAM* em GB nas 21 mil medições.

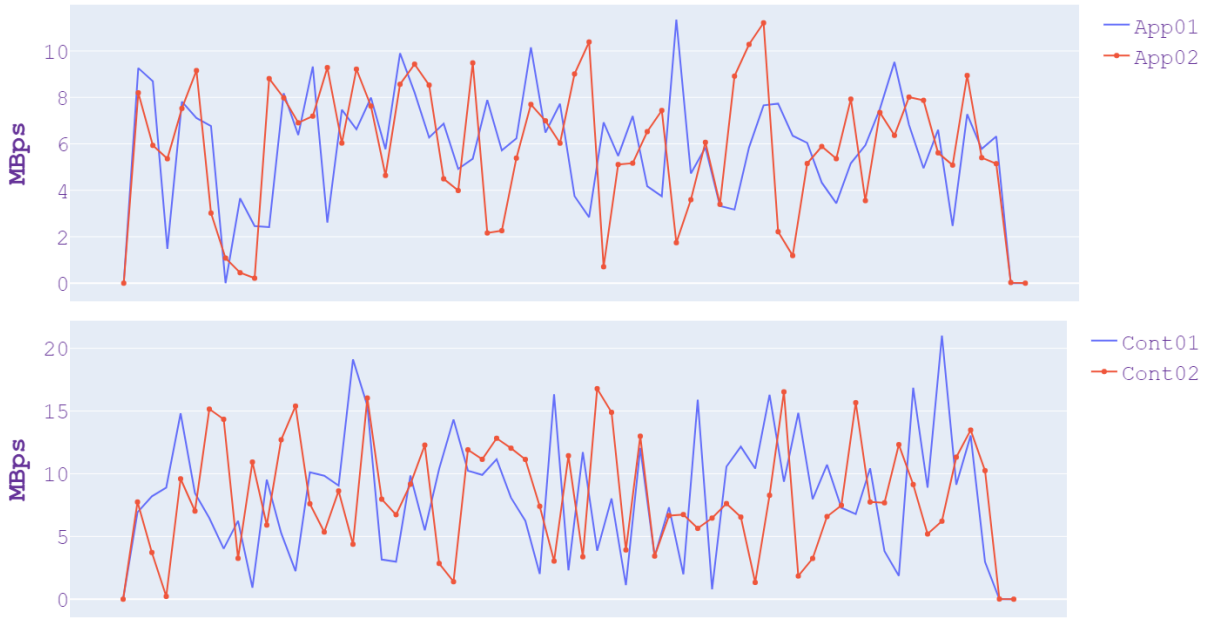
| | **App01** | **App02** | **Cont01** | **Cont02** |
|---|---|---|---|---|
| Média | 10.58 | 10.57 | 7.60 | 7.61 |
| Desvio Padrão | 0.34 | 0.33 | 0.27 | 0.27 |
| Mínimo | 9.98 | 9.97 | 7.14 | 7.15 |
| 25% | 10.29 | 10.28 | 7.37 | 7.37 |
| 50% | 10.58 | 10.57 | 7.61 | 7.61 |
| 75% | 10.87 | 10.86 | 7.84 | 7.83 |
| Máximo | 11.12 | 11.12 | 8.04 | 8.04 |

Fonte: Elaborado pelo autor (2023)

### 4.2.4 Uso de rede

Uma vez que o SIG@ é uma aplicação web, sobrecargas nas interfaces de rede podem ter resultados facilmente percebidos pelos usuários. A multiplexação dos pacotes realizada pelo hipervisor traz sobrecargas quando comparado a um servidor físico. Porém, uma aplicação conteinerizada tende a sofrer com instabilidades de rede, conforme descrito por Whiteaker *et al.* (2011).

Figura 29: Comparativo do uso rede nas *VMS* da camada de aplicação

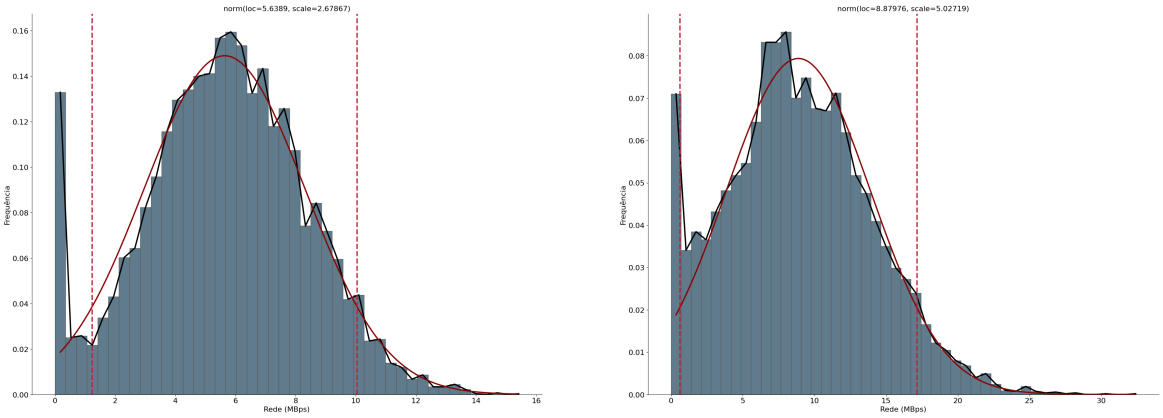



Fonte: Elaborado pelo autor (2023).

Os gráficos na Figura 29 mostram o fluxo agregado das interfaces de rede, a soma do tráfego que chega e do que sai de cada uma das VMs durante o primeiro teste em cada cenário.Durante este experimento, o cenário 1 apresentou um fluxo médio de 5,6 MBps, enquanto o cenário 2 apresentou um fluxo médio de 9 MBps. O comportamento em rajadas, comumente

Figura 30: Uso agregado de rede em *MBps* ajustado a curva normal

(a) App01 e App02 (b) Cont01 e Cont02

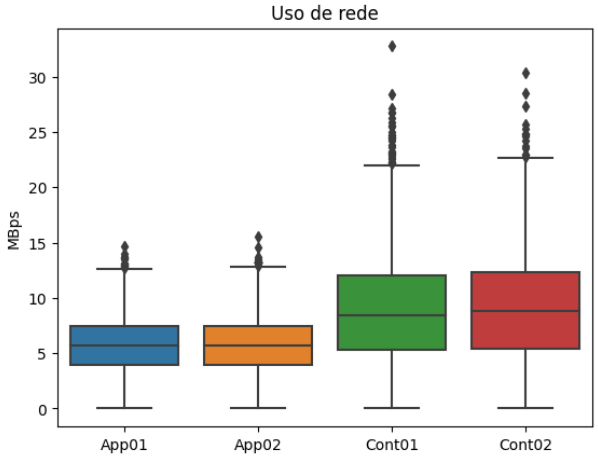



Fonte: Elaborado pelo autor (2023).

observado no monitoramento de redes, foi caracterizado em ambos os cenários devido ao desvio padrão das amostras ter sido superior à metade da média.

Na Figura 31, é possível visualizar os valores do tráfego de rede capturados nos cinco testes executados para cada cenário. É notável através da grande dispersão das medidas o comportamento em rajadas, comum em aplicações de rede, e também é possível observar que a versão conteinerizada do ambiente apresenta um fluxo de rede maior. Estes dados são dispostos na Tabela 9, onde podemos notar que o fluxo de dados no ambiente conteinerizado é em média 2.14 vezes maior do que no cenário tradicional. Na Figura 30 é notado o ajuste dos dados capturados nos cinco experimentos medidos à distribuição normal com parâmetros $\mu$ =5.64 e $\sigma$ = 2.68 no cenário 1 e $\mu$ =12.05 e $\sigma$ = 6.97 no cenário 2.

Figura 31: *Boxplot* do uso de rede das *VMS*.




Fonte: Elaborado pelo autor (2023).

Tabela 9: Uso de rede agregado em *MBps* nas 21 mil medições.

| | **App01** | **App02** | **Cont01** | **Cont02** |
|---|---|---|---|---|
| Média | 5.63 | 5.64 | 8.81 | 15.28 |
| Desvio Padrão | 2.64 | 2.71 | 4.99 | 8.95 |
| Mínimo | 0.0 | 0.0 | 0.0 | 0.0 |
| 25% | 3.92 | 3.90 | 5.30 | 5.40 |
| 50% | 5.71 | 5.66 | 8.44 | 8.79 |
| 75% | 7.44 | 7.48 | 12.04 | 12.31 |
| Máximo | 14.65 | 15.60 | 32.76 | 30.37 |

Fonte: Elaborado pelo autor (2023)

# 5

# CONCLUSÃO

Dada a sua capacidade de simplificar a implantação de aplicativos e melhorar a eficiência operacional em ambientes de computação em nuvem e de microsserviços, a conteinerização tem se tornado cada vez mais popular nos últimos anos. Este trabalho comparou o desempenho do SIG@ em máquinas virtuais e uma versão em contêineres executada em um ambiente que também utiliza máquinas virtuais, analisando o desempenho delas no uso do SIG@.

Assim, realizou-se uma análise prévia da estrutura, tendo como foco o período de maior criticidade e utilização do sistema, a matrícula, e criou-se uma carga de trabalho equivalente. Na análise de desempenho, notou-se que no cenário somente com VMs, 95,5% das requisições de matrícula foram atendidas com sucesso com tempo médio resposta de 2679 ms. No segundo cenário, contendo a versão conteinerizada da aplicação executada sobre máquinas virtuais, obteve-se sucesso de 95,6% das requisições de matrícula e tempo médio resposta de 2215 ms. Em termos de uso de recursos computacionais, o cenário com a aplicação conteinerizada apresentou um consumo inferior de memória RAM, porém de CPU e rede similares.

Os resultados mostraram que apesar da sobrecarga gerada pelo uso de contêineres sobre máquinas virtuais, é viável a implementação dessa arquitetura para a aplicação do SIG@, corroborando com a literatura. Logo, a adoção gradual ou completa de contêineres não deve afetar a qualidade do serviço prestado aos usuários finais e ainda trará vantagens, como a autoescalabilidade do serviço em tempo de execução.

## 5.1 TRABALHOS FUTUROS

Embora este trabalho tenha avançado na comparação dos cenários de implantação do SIG@, há possibilidades a serem exploradas tais como: aprimorar a carga de trabalho, simulando diferentes dispositivos teste, visando se aproximar da carga real registrada; relizar mais repeticões dos experimentos, dando mais confiabilidades aos resultados obtidos; avaliar o desempenho de um ambiente híbrido contendo um número reduzido máquinas virtuais com a aplicação não conteinerizada e outras máquinas virtuais sustentando um cluster autoescalável de contêineres gerenciado por uma ferramenta de orquestração atrelada ao balanceador carga do sistema. Esta arquitetura garante que o sistema esteja sempre acessível através da tecnologia utilizada há

anos no SIG@, mesmo que para um número reduzido de clientes, e à medida que o sistema for demandado novos containers são levantados para atender a carga extra.

# REFERÊNCIAS


Adams, K. & Agesen, O. (2006). A comparison of software and hardware techniques for x86 virtualization. *ACM Sigplan Notices*, 41(11):2–13.

al dhuraibi, Y., Fawaz, P., Djarallah, N., & Merle, P. (2017). Elasticity in cloud computing: State of the art and research challenges. *IEEE Transactions on Services Computing*, PP:1–1.

Analytics, G. (2023). Google analytics - sdk. `https://developers.google.com/analytics` [Acessado: 12/04/2023].

BlazeMeter (2023). Blazemeter chrome extension - record. `https://guide.blazemeter.com/hc/en-us/articles/13354685951505-Chrome-Extension-Record` [Acessado: 12/04/2023].

Clemente, D., Pereira, P., Dantas, J., & Maciel, P. (2022). Availability evaluation of system service hosted in private cloud computing through hierarchical modeling process. *The Journal of Supercomputing*, 78(7):9985–10024.

de Sousa Santos, I. & dos Santos Neto, P. d. A. (2008). Automaçao de testes de desempenho e estresse com o jmeter.

Docker (2023a). Docker overview. `https://docs.docker.com/get-started/` [Acessado: 12/04/2023].

Docker (2023b). Image layer details. `https://hub.docker.com/layers/library/ubuntu/jammy-20211122/images/sha256-3c3de9608507804525ff4303874525760ea36d62606e8105f515adaa761b80cb` [Acessado: 12/04/2023].

Dstat (2023). Dstat no red hat. `https://access.redhat.com/documentation/pt-br/red_hat_enterprise_linux/6/html/6.7_technical_notes/package-dstat` [Acessado: 12/04/2023].

Enberg, P. *et al.* (2016). A performance evaluation of hypervisor, unikernel, and container network i/o virtualization.

Felter, W., Ferreira, A., Rajamony, R., & Rubio, J. (2015). An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 171–172.

Hwang, J., Zeng, S., y Wu, F., & Wood, T. (2013). A component-based performance comparison of four hypervisors. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, 269–276.

Jin, Y., Wen, Y., Chen, Q., & Zhu, Z. (2013). An empirical investigation of the impact of server virtualization on energy efficiency for green data center. *The Computer Journal*, 56(8):977–990.

Jmeter (2023). Get started with jmeter. `https://jmeter.apache.org/usermanual/get-started.html` [Acessado: 12/04/2023].

Madeira, S. J. P. S. (2018). Avaliação de usabilidade do sistema de informações e gestão acadêmica da universidade federal de pernambuco (sig@/ufpe): um estudo de caso no centro de artes e comunicação. Master's thesis, Universidade Federal de Pernambuco.

Mavridis, I. & Karatza, H. D. (2017). Performance and overhead study of containers running on top of virtual machines. *2017 IEEE 19th Conference on Business Informatics (CBI)*, 02:32–38.

OpenLogic (2022). Ranking the top enterprise and open source operating systems of 2022. https://www.openlogic.com/blog/top-open-source-operating-systems-2022 [Acessado: 12/04/2023].

Oracle (2023). Oracle documentation library. https://docs.oracle.com/cd/E18283_01/index.htm [Acessado: 12/04/2023].

Rosenblum, M. & Garfinkel, T. (2005). Virtual machine monitors: Current technology and future trends. *Computer*, 38(5):39–47.

Sahoo, J., Mohapatra, S., & Lath, R. (2010). Virtualization: A survey on concepts, taxonomy and associated security issues. In *2010 second international conference on computer and network technology*, 222–226.

Shirinbab, S. & Lundberg, L. (2015). Performance implications of over-allocation of virtual cpus. In *2015 International Symposium on Networks, Computers and Communications (ISNCC)*, 1–6.

SIG@ (2023). Párgina inicial do siga. https://siga.ufpe.br/ufpe/index.jsp [Acessado: 12/04/2023].

Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., & Peterson, L. (2007). Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys european conference on computer systems 2007*, 275–287.

VMware (2023). Página do produto. https://www.vmware.com/br/products/vsphere.html [Acessado: 12/04/2023].

W3Techs (2023). Uso do apache segundo w3techs. https://w3techs.com/technologies/details/ws-apache [Acessado: 12/04/2023].

Whiteaker, J., Schneider, F., & Teixeira, R. (2011). Explaining packet delays under virtualization. *ACM SIGCOMM Computer Communication Review*, 41(1):38–44.

Zabbix (2023). Documentação oficial do zabbix. https://www.zabbix.com/manuals [Acessado: 12/04/2023].